JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

GRAPH-GRAMMATIKEN ZUR SUCHE UND KLASSIFIKATION VON MOLEKULAREN STRUKTUREN

Dissertation
zur Erlangung des Grades

"Doktor der Naturwissenschaften"

am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität
in Mainz

DOMENICO MOSCA

geboren am 27. März 1976
in Frankfurt am Main

Mainz, den 31. Juli 2019

TAG DER MÜNDLICHEN PRÜFUNG:
03. Dezember 2019

ABSTRACT

We define a graph rewriting system that is easily understandable by humans, but rich enough to allow very general queries to molecule databases. It is based on the substitution of a single node in a node- and edge-labeled graph by an arbitrary graph, explicitly assigning new endpoints to the edges incident to the replaced node. For these graph rewriting systems, we are interested in the subgraph-matching problem. We show that the problem is $\mathcal{NP}$-complete, even on graphs that are stars. As a positive result, we give an algorithm which is polynomial if both rules and the query graph have bounded degree and bounded cut size. We demonstrate that molecular graphs of practically relevant molecules in drug discovery conform with this property. The algorithm is not a fixed-parameter algorithm. Indeed, we show that the problem is $\mathcal{W}[1]$-hard on trees with the degree as the parameter.

Different approaches for learning graph grammars are presented. It is shown to what extent they can be used for substructure search or classification of molecules. Different aspects of production are considered, e.g., that a special structure is used, which was recognized in presented molecules. The results of the presented approaches are compared and evaluated with different machine learning approaches.

ZUSAMMENFASSUNG

In dieser Arbeit wird ein Modell zum Umschreiben von Graphen präsentiert, das sowohl einfach und intuitiv ist, aber mächtig genug, um allgemeine Abfragen in Moleküldatenbanken zu ermöglichen. Das Modell basiert auf der Ersetzung eines einzelnen Knotens in einem knoten- und kantengelabelten Graphen ohne Schleifen. Dies ermöglicht es, ein Modell vorzustellen, bei dem die Regeln vorgeben, wie die Kanten umgelenkt werden müssen, die einen Endpunkt verlieren, weil ein Knoten des Graphen entfernt wurde. Eine Regel ersetzt hierbei einen einzelnen Knoten mit einem bestimmten Label und deren inzidenten Kanten, die ebenfalls spezifische Label aufweisen, durch einen Teilgraphen. Hierbei spielt das Subgraphmatching eine entscheidende Rolle. Wir zeigen, dass das Problem auch auf Sterngraphen $\mathcal{NP}$-vollständig ist. Es wird ein Algorithmus vorgestellt, der eine polynomielle Laufzeit aufweist, falls die zugrundeliegenden Regeln einen begrenzten Grad aufweisen und der Anfragegraph eine begrenzte Größe des minimalen Schnittes inne hat. Wir zeigen, dass das Modell keinen FPT-Algorithmus darstellt und dass das Problem $\mathcal{W}[1]$-schwer auf Bäumen ist, mit dem Grad als Parameter.

Zusätzlich werden unterschiedliche Ansätze zum Lernen von Graph-Grammatiken vorgestellt. Anschließend wird aufgezeigt, in welchem Umfang diese zur Substruktursuche oder Klassifizierung von Molekülen genutzt werden können und welche Ergebnisse sie hierbei erzielen. Die Ergebnisse der vorgestellten Ansätze werden mit unterschiedlichen Ansätzen des maschinellen Lernens miteinander verglichen und bewertet.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

AKRONYME

BPKP beschränktes Postsches Korrespondenzproblem

DFS Tiefensuche

DP Dynamische Tabelle

DPO Double-Pushout-Approach, dt. Ansatz mit doppeltem Pushout

FPT Fixed-Parameter-Algorithm, dt. parametrisierter Algorithmus

GG Graph-Grammar, dt. Graph-Grammatik

HRG Hyperedge Replacement Graph-Grammar, dt. Hyperkantenersetzende Graph-Grammatik

INCHI IUPAC International Chemical Identifier

LCSP Longest-Common-Subsequence-Problem, dt. Längstes gemeinsames Subsequenz Problem

NLC Node-Label-Controlled-Approach, dt. Knotenlabel kontrollierter Ansatz

NRG Node Replacement Graph-Grammar, dt. Knotenersetzende Graph-Grammatik

SLN Sybyl Line Notation

SMARTS SMILES arbitrary target specification

SMILES Simplified Molecular Input Line Entry Specification

SPO Single-Pushout-Approach, dt. Ansatz mit einzelnem Pushout

SSSR Smallest Set of Smallest Rings, dt. Kleinste Menge an kleiner Ringe

USA United States of America, dt. Vereinigten Staaten von Amerika

XP Slice-wise polynomial

1 EINFÜHRUNG

Kleine Moleküle [1] spielen in der Molekularbiologie eine bedeutende Rolle. Moleküle sind elektrisch neutrale Teilchen, die aus zwei oder mehr Atomen bestehen [94]. Sie können unterschiedliche Funktionen einnehmen, wie z. B. als Inhibitoren oder Aktivatoren von Proteinen, als Informationsträger oder als energetische Speicher. Kleine Moleküle sind daher Schwerpunkt zahlreicher Forschungsgebiete ([4, 35, 36, 39]). Besonders in der Arzneimittelforschung werden kleine Moleküle dazu verwendet, um Proteine zu hemmen oder zu aktivieren, eine gewünschte biologische Funktion zu erreichen, oder um unerwünschte zu verhindern. Ein Hemmstoff verlangsamt, hemmt oder verhindert chemische Reaktionen [65]. Aktivatoren sind Verbindungen, welche die Wirkung eines Katalysators beschleunigen [66]. Katalysatoren sind Stoffe, die die Geschwindigkeit einer chemischen Reaktion durch Herabsetzung der Aktivierungsenergie erhöhen [115]. Oft sind bestimmte Molekülsubstrukturen entscheidend für die Entwicklung unterschiedlicher Aspekte ihrer biologischen Funktion, z. B. können sie eine toxische Wirkungen hervorrufen, andere wiederum das gewünschte Ergebnis erzeugen. Daher stellt die Erforschung von molekularen Substrukturen eine der wichtigsten Aufgaben im Bereich der Chemoinformatik, Biologie und Pharmazie dar [84, 88, 91, 125]. Aus diesem Grund ist die Repräsentation von Substrukturen eine Kernkomponente der meisten Anwendungen in der Chemoinformatik, einschließlich der Datenbankrecherche oder des virtuellen Screenings [48, 123].

Um Moleküle effektiv nach Substrukturen durchsuchen zu können, ist eine geeignete Repräsentation notwendig. Im Allgemeinen werden Substrukturen mittels chemischer Nomenklaturen wie die der Daylight's SMARTS modelliert. Es existiert eine Vielzahl unterschiedlicher Repräsentationen. Diese sind in der Regel sehr komplex für den Anwender und erfordern einen hohen Einarbeitungsaufwand, bevor sie routinemäßig eingesetzt werden können. Die derzeit bekanntesten Nomenklaturen sind die SYBYL Line Notation (SLN), Simplified Molecular Input Line Entry Specification (SMILES) und SMILES arbitrary target specification (SMARTS) [50]. Die vorhandenen Nomenklaturen unterstützen einige, aber nicht alle relevanten Anforderungen, wie zum Beispiel die Abfrage von zwei beliebigen, aber identischen molekularen Substrukturen im Molekül. Auch fehlt es diesen chemischen Beschreibungssprachen häufig an formalen Semantikdefinitionen. Darüber hinaus sind diese Sprachen sehr eingeschränkt in ihrer Fähigkeit, Pattern von Typologien der zugrunde liegenden Graphen

zu beschreiben. Dies ist z. B. bei der Suche nach bestimmten zyklischen Peptiden relevant. Viele wichtige Medikamente weisen solch eine Struktur auf, da sie dazu neigen, der Verdauung recht effektiv zu widerstehen. Die molekulare Funktion beruht dabei auf molekularen Wechselwirkungen. Arzneimittelwechselwirkungen (auch Arzneimittelinteraktionen) können bei gleichzeitiger Einnahme verschiedener Arzneimittel auftreten. Wechselwirkungen erfordern typischerweise kein Zusammenspiel von allen Atomen eines Moleküls , sondern nur von bestimmten Atomgruppen. Die Wirkung eines Moleküls lässt sich folglich nicht auf die gesamte Molekülstruktur, sondern auf eine Substruktur dieses Moleküls zurückführen. Daher ist davon auszugehen, dass Moleküle, die einen bestimmten molekularen Subgraphen aufweisen, auch einige ihrer Wechselwirkungen und damit Funktionen innehaben.

Das Suchen und Finden von chemischen Substrukturen in einer Datenbank, die eine gewünschte medizinische Wirkung hervorrufen, wird in dieser Arbeit unter Zuhilfenahme von Graph-Grammatiken erfolgen. Die Schwierigkeit liegt darin, dass das Parsen und Zuordnen von molekularen Substrukturen im Molekül eine sehr komplexe und zeitintensive Abfrage in Datenbanken darstellt. Dieses Suchproblem ist $\mathcal{NP}$-vollständig.

Um diese Probleme zu umgehen, wird in dieser Arbeit ein einfaches Graphersetzungssystem, welches Substrukturen beschreiben kann, präsentiert. Die Verwendung von Methoden zur direkten Beschreibung von Graphen im Vergleich zu einer chemischen Nomenklatur intuitiver ist. Bereits triviale Graphersetzungssysteme ermöglichen eine ähnlich hohe Ausdruckskraft wie die der SMARTS. Zusätzlich ermöglichen sie es Substrukturen zu definieren, die diese sogar übersteigen. Obgleich das damit verbundene Suchproblem $\mathcal{NP}$-vollständig bleibt, wird es polynomiell, falls jeder minimale Schnitt des Anfragegraphen eine begrenzte Größe aufweist. Dies gilt empirisch für die meisten in den Standarddatenbanken enthaltenen Moleküle. Die Erfahrung hat gezeigt, dass die Verwendung von Graph-Grammatiken anstelle molekularer Beschreibungssprachen ermöglicht nicht nur eine intuitivere Darstellung in der Graphentopologie, sondern auch Abfragen, die nicht z. B. mittels SMARTS realisiert werden können [4]. Typischerweise ist dies nur durch zusätzliche externe Filter möglich. Funktionelle Gruppen können durch Graph-Grammatiken intuitiver beschrieben werden als mit chemischen Nomenklaturen, indem sie genau die molekularen Subgraphen, die zur Klasse der jeweiligen Gruppe gehören, erzeugen. Das Matchen eines solchen Graph-Ersetzungssystems in einer molekularen Datenbank ermöglicht es daher, nach Instanzen zu suchen, die eine bestimmte chemische Gruppe enthalten und somit möglicherweise die gewünschte chemische Funktion aufweisen. Ein Graph-

Ersetzungssystem wird verwendet, um eine Sprache von Graphen zu definieren, ähnlich wie bei bekannten Nomenklaturen. Daher werden Graph-Ersetzungssysteme auch als „Graph-Grammatiken“ bezeichnet. Ein Graph-Ersetzungssystem besteht aus einer Menge an Regeln, die definieren, wie ein Graph in einen anderen umgewandelt werden kann. Die entsprechende Sprache ist die Menge aller Graphen, die aus den Regeln abgeleitet werden kann. Am Anfang besteht der Graph aus einem einzelnen Knoten mit einem bestimmten Label. Es existieren mehrere bekannte Definitionen zu Graphersetzungssystemen. Diese sind z. B. die Knotenersetzenden-, Hyperkantenersetzenden- oder die Single- und Double-Pushout-Ansätze, die sich in der Art der Transformationsregeln unterscheiden. Dabei wird eine Definition eines Graphersetzungssystems präsentiert, das einfach und zugleich intuitiv ist, da die Graphen, die hier betrachtet werden, von beschränktem Grad sind (Atome in Molekülen können nur eine endliche und typischerweise sehr geringe Anzahl von kovalenten Bindungen eingehen). Aus diesem Grund weisen die vorgestellten Ersetzungssysteme einen beschränkten Grad auf. Dies ermöglicht es in dieser Arbeit ein Modell vorzustellen, bei dem die Regeln vorgeben, wie die Kanten umgelenkt werden müssen, die einen Endpunkt verlieren, weil ein Knoten des Graphen entfernt wurde. Graphen, die in dieser Arbeit genutzt werden, sind knoten- und kantengelabelte Multigraphen ohne Schleifen. Eine Regel ersetzt hierbei einen einzelnen Knoten mit einem bestimmten Label und deren inzidenten Kanten, die ebenfalls spezifische Label aufweisen durch einen Teilgraphen. Kanten, die ursprünglich zum entfernten Knoten inzident waren, erhalten im neuen Teilgraphen neue Endpunkte. Dabei dürfen die Label dieser Kanten, falls sie bereits ein festes Label aufwiesen, nicht mehr verändert werden.

1.1 ZIEL

Diese Arbeit soll aufzeigen, dass ein Graph-Grammatik-Ansatz zur Substruktursuche verwendet werden kann. Das Ziel ist es, einen Algorithmus zur Substruktursuche oder Klassifikation von Molekülen vorzustellen, der sowohl einfach als auch intuitiv ist. Er soll auch sehr allgemeine Datenbankabfragen ermöglichen. Beim Wirkstoffdesign wird in einer Datenbank eine möglichst hohe Anzahl an ähnlichen Strukturen gesucht. Eine Moleküldatenbank kann eine Vielzahl von Einträgen aufweisen, so dass in der Regel viel Zeit benötigt wird, um Treffer einer bestimmten chemischen Substruktur oder chemischen Gruppe aufzufinden. Des Weiteren wird die Komplexität des vorgestellten Algorithmus untersucht und unter welchen Voraussetzungen eine polynomielle Laufzeit erreicht werden kann. Ebenfalls werden unterschiedliche Ansätze zum Erlernen von Graph-Grammatiken vorgestellt, die zur Substruktursuche oder Klassifikation verwendet wer-

den können. Diese werden später mit Ansätzen des maschinellen Lernens miteinander verglichen. Hierbei soll untersucht werden, welche der vorgestellten Ansätze zur Klassifikation besser geeignet sind.

1.2 AUFBAU

Kapitel 2 gibt es einen Überblick über die formalen Grundlagen. Hierbei werden anfangs Repräsentationen chemischer Nomenklaturen vorgestellt. Als erstes wird die SMILES-Notation in Abschnitt 2.1.1, einen Abschnitt später in 2.1.2 die Erweiterung der SMILES-Notation, die SMARTS-Notation eingeführt. Unique SMILES werden in Abschnitt 2.1.3 eingeführt. In Abschnitt 2.1.4 erhält der Leser einen kurzen Einblick in die SLN-Notation. Sie kann als eine Erweiterung der SMILES Notation angesehen werden [4]. Da in der Chemoinformatik Moleküle in der Regel als Graphen dargestellt werden, folgen in Abschnitt 2.2 Definitionen aus der Graphentheorie. Kernthema dieser Arbeit ist die Entwicklung eines Algorithmus unter Zuhilfenahme von Graph-Grammatiken zur Substruktursuche, daher werden in Abschnitt 2.3 unterschiedliche Graphersetzungssysteme vorgestellt. Sie fungieren als formale Beschreibung bei der Veränderung von Graphen. Eine Einführung in die Komplexitätstheorie wird in Abschnitt 2.4 erfolgen. Die Komplexitätstheorie klassifiziert die Komplexität von Problemen im Hinblick auf den notwendigen Aufwand um eine gewünschte Lösung in einer bestimmten Zeit zu erhalten. In Abschnitt 2.5 der formalen Grundlagen erfolgt eine Einführung in das maschinelle Lernen. Das maschinelle Lernen wird in dieser Arbeit in Kapitel 9 zum Klassifizieren von Molekülen genutzt. In Abschnitt 2.5.1 werden künstlich neuronale Netze, in Abschnitt 2.5.2 rekurrente neuronale Netzwerke und in Abschnitt 2.5.3 der Random Forest Ansatz eingeführt. Kapitel 3 stellt die bekanntesten Ansätze zur Substruktursuche vor. Zum einen wird der Algorithmus von Ullmann [117] in Abschnitt 3.1 und zum anderen der VF2-Algorithmus von Cordella et al. [26] in Abschnitt 3.2 dargestellt. Beide Algorithmen beruhen auf einem Backtrackingansatz. In Kapitel 4 werden Definitionen zum Algorithmus zur Substruktursuche eingeführt. In Kapitel 5 wird erläutert wie Graph-Grammatik-Regeln normalisiert werden, um die Beschreibung des Algorithmus aus Kapitel 6 zu vereinfachen. Kapitel 6 wird den entwickelten Algorithmus vorstellen. Hierbei wird erläutert, wie Graph-Grammatiken zur Substruktursuche genutzt werden können. Ebenfalls wird aufgezeigt, wie die zu berücksichtigenden Teilmengen bei der Substruktursuche reduziert werden können. Das Kapitel 7 klassifiziert die Komplexität des Ansatzes. Es wird gezeigt, dass das Problem auf Bäumen sowohl $\mathcal{NP}$-vollständig (Abschnitt 7.1) als auch $\mathcal{W}[1]$-schwer ist (Abschnitt 7.2). In Abschnitt 7.3 wird gezeigt, dass das Problem auch auf Bäumen mit beschränktem Knotengrad und beschränkter Baumweite

$\mathcal{NP}$-vollständig bleibt. Abschnitt 7.4 zeigt auf, wie die Größe einer dynamischen Tabelle für eine Graphenklasse, die durch einen bestimmten Parameter beschränkt worden ist, abgeschätzt werden kann.

In Kapitel 8 werden unterschiedliche Ansätze zur Erzeugung von Graph-Grammatiken vorgestellt. Anschließend wird aufgezeigt, in welchem Umfang diese zur Substruktursuche oder Klassifizierung von Molekülen genutzt werden können und welche Ergebnisse sie erzielen. Dabei werden unterschiedliche Aspekte bei der Erzeugung betrachtet, wie z. B. dass eine spezielle Struktur genutzt wird, die in Molekülen erkannt wurde. In Abschnitt 8.6 folgen Ergebnisse zur Nutzung von Graph-Grammatiken zur Substruktursuche, die unter Zuhilfenahme der Verfahren aus Kapitel 8 berechnet wurden. Die Ergebnisse der vorgestellten Ansätze aus Kapitel 8 werden mit den Ansätzen aus dem Kapitel des maschinellen Lernens in den Abschnitten 9.1.4 und 9.2.2 miteinander verglichen und bewertet. Genauer gesagt, unter Zuhilfenahme rekurrenter neuronaler Netze (RNN) und des Random Forest Ansatzes (dt. Zufallswälder) wird versucht, Moleküle in unterschiedliche funktionale Klassen einzuordnen. Kapitel 10 fasst die Erkenntnisse zusammen, gibt ein kritisches Fazit und einen Ausblick auf mögliche Erweiterungen.

2 FORMALE GRUNDLAGEN

In diesem Kapitel werden die formalen Grundlagen eingeführt, beginnend mit einem kurzen Überblick über die Geschichte der chemischen Nomenklaturen und der unterschiedlichen Darstellungsweisen von Molekülen im 19. Jahrhundert. Später folgt eine Einführung in die aktuell von Forschern meistgenutzten chemischen Repräsentationsarten, um Moleküle zu beschreiben. Die derzeit meistgenutzten chemischen Repräsentationsarten sind die SMILES- und SMARTS-Notationen der Firma Daylight Chemical Information Systems und die SYBYL Line Notation (SLN) [50, 70, 101, 120, 121].

2.1 REPRÄSENTATION CHEMISCHER STRUKTUREN

Als die Chemie im 18. Jahrhundert aus der Alchemie hervorging, wurden einfache Nomenklaturen zur Identifizierung chemischer Substanzen genutzt. Erst gegen Ende des 19. Jahrhunderts haben sich systematische Nomenklaturen etabliert. In den Jahren nach dem Zweiten Weltkrieg wurden eine Reihe von Linearnotationen entwickelt, die eine kompakte Darstellung der chemischen Strukturen ermöglichen. Linearnotationen konnten bereits chemische Substrukturen in einem Molekül, wie z. B. funktionale Gruppen und Ringsysteme, erkennen. Die ersten Linearnotationen wurden vor dem Computerzeitalter entwickelt. Damals war der Computerspeicher sehr teuer und knapp. Dies war der Grund, warum sich Linearnotationen zum damaligen Zeitpunkt durchgesetzt haben. Der Computerspeicher wurde im Laufe der Jahre immer günstiger, somit konnten Moleküle ausführlicher dargestellt und Atome und Bindungen tabellarisch aufgelistet werden. Die tabellarische Beschreibung von chemischen Molekülen war bis Anfang der 1980er Jahre die bestimmende Darstellungsart. Tabellarische Darstellungen ließen aber keine rasche Verarbeitung der chemischen Strukturen durch den Menschen zu. Aus diesem Grund wurden andere Verfahren konzipiert und weiterentwickelt. Dies hat dazu geführt, dass Linearnotationen, die aus den 1940er und 1950er Jahren stammen, wieder Beachtung fanden [50].

Die heutigen Linearnotationen bestehen aus alphanumerischen Strings (dt. alphanumerischen Zeichenketten). Die heute am häufigsten genutzten Linearnotationen sind die SMILES-, SMARTS- und die SLN-Notation. Soweit nicht anders angegeben, basieren die Abschnitte 2.1.1, 2.1.2 und 2.1.3 auf der Arbeit von Weiniger et al. [121].

2.1.1 *SMILES-Notation*

Die SMILES-Notation wurde in den späten 1980er Jahren von David Weininger [120, 121] entwickelt. Die Firma Daylight Chemical Information Systems Inc. [73] entwickelte diesen Ansatz in den folgenden Jahren weiter. Die SMILES-Notation wird zur Beschreibung von Molekülen und chemischen Reaktionen genutzt. SMILES ist eine chemische Nomenklatur, die Moleküle in vereinfachter Form als Strings darstellt. SMILES stellen den Aufbau eines Moleküls samt der vorkommenden Atome sowie deren Bindungen dar. Hierbei wird ein Molekül durch einen Graphen präsentiert, wobei Knoten als Atome und Kanten als Bindungen eines Moleküls dargestellt werden. Nachfolgend werden die Konventionen zur SMILES-Notation erläutert [74].

Atome werden im SMILES-String mit ihrem chemischen Elementsymbol dargestellt. Zur Darstellung von Bindungen existieren unterschiedliche Symbole in der SMILES-Notation. Einige werden explizit geschrieben, andere implizit mitgelesen. Mehrfachbindungen, wie Doppel- oder Dreifachbindungen werden explizit, Einfachbindungen implizit dargestellt. Die SMILES-Notation ermöglicht es auch Zyklen unterschiedlicher Länge zu beschreiben. Der Anfang und das Ende eines Zyklus in einem Molekül werden in SMILES mittels Ringzahlen dargestellt. Atome, die in einem Molekül mehr als drei Verbindungen aufweisen, werden unter Zuhilfenahme von Branches (dt. Verzweigungen) notiert. Branches werden in der SMILES-Notation mit runden Klammern angegeben. SMILES und später SMARTS in Abschnitt 2.1.2 verwenden größtenteils eine gemeinsame Regelmenge, um Strukturen durch einen String zu beschreiben.

2.1.1.1 *Atome*

Atome werden durch ihre chemischen Elementarsymbole in eckigen Klammern dargestellt. Ein Kohlenstoffatom wird mit [C], Chlor mit [Cl] und Sauerstoff mit einem [O] beschrieben. Zur Vereinfachung können die eckigen Klammern bei der organischen Teilmenge weggelassen werden. Elemente der organischen Teilmenge sind B, C, N, O, P, S, F, Cl, Br und I. Das erste Symbol eines chemischen Elementes wird in der SMILES-Notation immer groß, die Folgenden klein geschrieben, außer es liegen aromatische Atome vor, diese beginnen mit einem kleinen Buchstaben wie z. B. einem aromatischen Kohlenstoffatom [c].

Ladungen werden durch Angabe einer Ladungszahl am Ende der eckigen Klammer notiert `[Cl-2]`. Die Ladungszahl kann sowohl positive wie auch negative Werte annehmen ([Atom ± Ladungszahl]). Wasserstoffatome (H-Atome) werden in der SMILES-Notation implizit notiert, d. h. sie werden nicht im SMILES-String aufgeführt. So

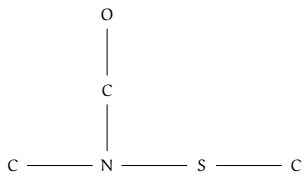


Abbildung 1: Branches in SMILES

Der SMILES-String CN(CO)SC wird am N Atom verzweigt, da das Stickstoffatom mehr als zwei Verbindungen zu anderen Atomen in Molekülen aufweist. Der Nebenast besteht aus den Elementen CO. Die Atome CNSC sind im Molekül miteinander verbunden, die Atomkette CO nur mit dem Stickstoffatom (dieses Molekül dient nur der Veranschaulichung und stellt eine Substruktur der Triflumidate dar [95]).

besitzt der SMILES-String für Methan nur das C Atom. Die vier angebundenen Wasserstoffatome werden implizit mitgelesen. Die an das Kohlenstoffatom direkt angebundenen Wasserstoffatome können aber auch explizit geschrieben werden. Hierbei wird das Kohlenstoffatom in eckige Klammer gesetzt und die vier Wasserstoffatome hinten angestellt [CH4].

Werden die eckigen Klammern nicht notiert, stimmt die Anzahl an gebundenen Wasserstoffatomen mit der niedrigsten normalen Wertigkeit des Elements überein. Die Ladung des Elements ist gleich null und das Atom besitzt keine zusätzliche Beschreibungen. Die Syntax für Atome lautet: `[Elementsymbol<#H-Atome><Vorzeichen<Ladung»]`.

2.1.1.2 *Bindungen*

Nachfolgend werden die in der SMILES-Notation definierten Bindungsarten vorgestellt.

- Die Einfachbindung wird durch das Minuszeichen (−),
- die Doppelbindung durch das Gleichheitszeichen (=),
- die Dreifachbindung durch das Rautezeichen (#) und
- die aromatische Bindung durch den Doppelpunkt (:) dargestellt.

Eine Einfachbindung (−) wird, falls kein anderes Bindungssymbol notiert ist, implizit gelesen, kann aber auch explizit geschrieben werden. Für die folgende Kohlenstoffkette sind zwei unterschiedliche Schreibweisen möglich:

- `C-C`,
- `CC`.

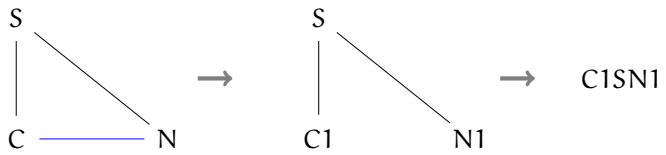


Abbildung 2: Zyklen in SMILES
Im linken Molekül wird die Bindungskante zwischen den Kohlenstoff- (C) und Stickstoffelementen (N) gelöscht. Es wird jedem Atom, mit der die Kante verbunden war, eine Ringzahl hinzugefügt, hier die Eins. Daraus folgt der SMILES-String C1SN1 (oder N1SC1).

2.1.1.3 *Branches*

Besitzt ein Atom im Molekül mehr als drei Verbindungen, werden Branches (dt. Verzweigungen) genutzt. Branches werden in SMILES mit runden Klammern dargestellt ( ). Sie können ineinander verschachtelt oder übereinander angeordnet werden. Ein Branch gehört immer zu dem links außerhalb der runden Klammer notierten Atom. Abbildung 1 verdeutlicht das Vorgehen in der SMILES-Notation.

2.1.1.4 *Zyklen*

Zyklische Molekülstrukturen werden in der SMILES-Notation wie folgt dargestellt. Man entfernt dem Zyklus an einer beliebigen Stelle eine Bindungskante und fügt an den Atomen, mit der die Bindungskante verbunden war, jeweils die gleiche Zahl, aber eine beliebige Ringzahl an (siehe Abbildung 2).

Jedem Element können mehrere Ringzahlen gleichzeitig zugeordnet werden. Ringzahlen, die größer als neun sind, werden zusätzlich mit einem Prozentzeichen (%) markiert. Befindet sich nach einem chemischen Element eine Zahl, die aus mehr als einer Ziffer besteht, und keine Ziffer dieser Zahl ist zusätzlich mit einem (%) Zeichen markiert, so ist jede Ziffer am Atom genau eine Zyklusbindung. Zwei Atome können untereinander nicht mehr als eine Bindung aufweisen und ein Atom kann nicht mit sich selbst verbunden sein.

C23%10ON[Cl]4%10SCCC2C3C4

Der obige SMILES-String weist am ersten Kohlenstoffatom (roter Teil) drei und am Chloratom (blauer Teil) genau zwei Zyklusbindungen auf. Die graphische Darstellung des obigen SMILES-Strings ist in Abbildung 3 dargestellt.

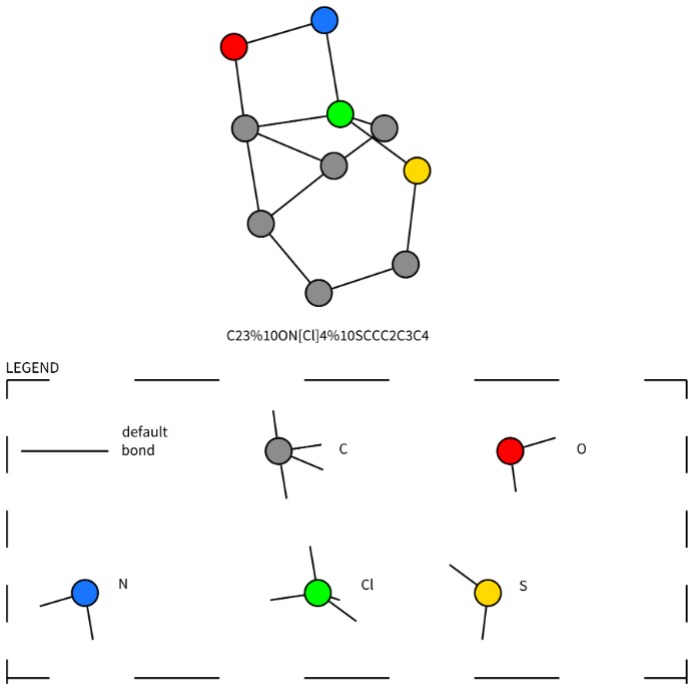


Abbildung 3: Graphische Darstellung eines SMILES-Strings
Die Abbildung zeigt die graphische Darstellung des SMILES-Strings C23%10ON[Cl]4%10SCCC2C3C4 [13].

2.1.1.5 *Nicht zusammenhängende Moleküle*

Nicht zusammenhängende Moleküle können ebenfalls mit einem einzelnen SMILES-String beschrieben werden. Die Moleküle werden in diesem Fall durch einen Punkt (.) voneinander getrennt.

CCC.CCC

SMARTS ermöglichen es Substrukturen zu definieren. Sie können zusätzlich zu SMILES genau definierte Atom- und Bindungssymbole aufweisen. Der nächste Abschnitt definiert die zusätzlichen Notationen [73].

2.1.2 *SMARTS-Notation*

Die SMARTS-Notation ist wie die SMILES-Notation eine chemische Nomenklatur, die Moleküle in vereinfachter Form als Strings darstellen kann. Sie stellt eine Erweiterung der SMILES-Notation dar und erweitert die SMILES-Notation um zusätzliche Beschreibungen, dadurch wird es ermöglicht Atome und Bindungen genauer zu beschreiben.

Im Gegensatz zur SMILES-Notation kann die SMARTS-Notation in der Chemoinformatik zur Substruktursuche genutzt werden. In vielen praktischen Anwendungen der Chemoinformatik zur molekularen Beschreibungen werden sie verwendet.

Nachfolgend werden die wichtigsten Konventionen zur SMARTS-Notation erläutert. Sie stellt ebenso wie die SMILES-Notation den Aufbau eines Moleküls samt der vorkommenden Atome sowie deren Bindungen dar. Zur Darstellung von Bindungen werden zusätzlich zu den in der SMILES-Notation eingeführten Konventionen weitere Bindungsarten genutzt.

2.1.2.1 *Atome*

Wie in der SMILES-Notation werden SMARTS-Atome mit ihren chemischen Elementsymbolen beschrieben. Atome können aber alternativ mit der Ordnungszahl dargestellt werden. Das Element wird hierbei in einer eckigen Klammern notiert. Vor der Ordnungszahl steht das Rautesymbol. Die folgenden Beschreibungen für ein Kohlenstoffatom sind in der SMARTS-Notation äquivalent [#6] $\Leftrightarrow$ [C]. Zusätzlich wird das Wildcard-Symbol ($*$) eingeführt. Dieses dient z. B. als Platzhalter für ein beliebiges Atom im Molekül.

Es existieren Symbole, die die Atomeigenschaft genauer definieren wie z. B. Ladung, Valenz und Grad. Zusätzlich existiert die Möglichkeit Atome allgemeiner zu beschreiben.

Atome können zusätzliche Attribute aufweisen. Die Syntax für ein SMARTS-Atom ist zum Schutz vor Ambiguitäten wie folgt definiert: [[Masse][Elementsymbol][Ringzahl][zusätzliche Eigenschaften]].

Der SMARTS-String [C−] beschreibt z. B. ein negativ geladenes Kohlenstoffatom. Die zusätzliche Eigenschaft eines Atoms ist über ihre Position in der Notation im SMARTS-String beschrieben.

Die Bezeichnung für die Atommasse wird am Anfang direkt nach der eckigen Klammer beschrieben und steht vor dem Elementsymbol. Ringzahlen werden direkt nach dem Elementsymbol spezifiziert. Ein aromatisches Atom wird mit einem kleinen a, ein aliphatisches hingegen mit einem großen A notiert. Moleküle, die nur Kohlenstoff (C-Atome) und Wasserstoff (H-Atome) enthalten, werden Kohlenwasserstoffverbindungen genannt. Ein Molekül heißt topologisch chiral, falls es nicht durch Umformungen in einem gegebenen Raum in sein Spiegelbild überführt werden kann [23]. Sei $< n >$ eine Zahl und $< c >$ eine chirale Klasse [120]. In Tabelle 1 erhält man eine Übersicht über die in der SMARTS-Notation verwendeten Symbole zur erweiterten Beschreibung von Atomen.

Die Ladung eines Atoms kann auf zwei Arten erfolgen. Entweder durch die Anzahl angegebener Vorzeichen ([C − −]) oder es wird ein Vorzeichen, gefolgt von einer Zahl notiert ([C − 2]). Beide Schreibweisen sind in diesem Fall wieder äquivalent und beschreiben ein Kohlenstoffatom mit einer negativen Ladung von 2. Es existieren zusätzliche Atombezeichner. Diese sind D, X, H, h, v, R, r und sind wie folgt definiert.

- D: Anzahl explizit adjazenter Atome; dabei ist die Bindungsart belanglos; implizite Wasserstoffatome werden nicht mitgezählt
- X: Anzahl adjazenter Atome; unabhängig von der Bindungsart; implizite Wasserstoffatome werden mitgezählt
- H: Anzahl adjazenter Wasserstoffatome
- h: Anzahl impliziter adjazenter Wasserstoffatome
- v: gibt die Valenz an; beschreibt die Anzahl an inzidenten Bindungen
- R: Anzahl an Smallest Set of Smallest Rings (SSSR, dt. Kleinste Menge an kleinsten Zyklen) in dem sich das Atom befindet
- r: Größe des SSSR in dem sich das Atom befindet

2.1.2.2 *Bindungen*

Die SMARTS-Notation erweitert die in SMILES eingeführten Bindungsarten. Neu in der SMARTS-Notation ist die Möglichkeit gerichtete und beliebige Bindungen darzustellen. Nachfolgend werden die in SMARTS eingeführten Bindungsarten beschrieben. Tabelle 2 stellt alle in SMARTS gültigen Bindungsarten dar.

2.1.2.3 *Branches und Zyklen in SMARTS*

Die SMARTS-Notation nutzt zur Darstellung von Branches und Zyklen die gleiche Semantik wie die SMILES-Notation aus den Abschnitten 2.1.1.3 und 2.1.1.4.

2.1.3 *Unique SMILES*

SMILES beschreiben Molekülstrukturen mittels Strings (siehe Abschnitt 2.1.1). Wie in Abbildung 2 dargestellt, existieren in der SMILES-Notation unterschiedliche Arten, das gleiche Molekül zu beschreiben. Es besteht jedoch die Möglichkeit aus allen gültigen Möglichkeiten einen speziellen SMILES zu erzeugen. Unter Zuhilfenahme eines Kanonisierungsalgorithmus wird ein gültiger und einzigartiger

Tabelle 1: SMARTS – erweiterte Atombezeichner
Die Tabelle gibt die unterschiedlichen Erweiterungsmöglichkeiten an, um die zusätzlichen atomaren Eigenschaften in einem SMARTS zu beschreiben. Hierbei steht <n> z. B. für eine Ziffer und <c> für eine chirale Klasse.

Symbol	Symbolname	Atom Eigenschaften	Standard
*	Wildcard	beliebiges Atom	–
a	aromatisch	aromatisch	–
A	aliphatisch	aliphatisch	–
D < n >	Grad	< n > explizite Verbindungen	exakt eine
H < n >	Gesamtanzahl an H Atomen	< n > gebundene H-Atome	exakt eins
h < n >	implizite Anzahl H	< n > implizite H-Atome	exakt eins
R < n >	Ringzugehörigkeit	< n > SSSR Ringe	beliebiges Ringatom
r < n >	Ringgröße	im kleinsten SSSR Ring der Größe < n >	beliebiges Ringatom
v < n >	Valenz	Gesamt Bindungsordnung < n >	exakt eins
X < n >	Konnektivität	< n > gesamte Anzahl Bindungen	exakt eins
− < n >	negative Ladung	− < n > Ladung	negative Ladung −1
+ < n >	positive Ladung	+ < n > Ladung	positive Ladung von +1
#n	Atomnummer	Atomnummer < n >	–
@	Chiralität	gegen Uhrzeigersinn	gegen Uhrzeigersinn, Standardklasse
@@	Chiralität	im Uhrzeigersinn	im Uhrzeigersinn, Standardklasse
@ < c >< n >	Chiralität	chirale Klasse < c > Chiralität < n >	–
@ < c >< n >?	chiral oder unbestimmt	Chiralität < c >< n > oder unbestimmt	–
< n >	Atommasse	explizite Atommasse	unbestimmte Masse

Tabelle 2: SMARTS – Bindungen
Alle Bindungstypen, die in der SMARTS-Notation genutzt werden, sind in der obigen Tabelle aufgeführt. Das (@)-Zeichen z. B. gibt eine beliebige Ring- und das (~)-Symbol eine aromatische Bindung an [120].

Symbol	Symbolname
–	Einfachbindung (aliphatisch)
/	nach oben gerichtete Einfachbindung
\	nach unten gerichtete Einfachbindung
/?	Richtungsbindung "nach oben oder unspezifiziert"
\?	Richtungsbindung "nach unten oder unspezifiziert"
=	Doppelbindung
#	Dreifachbindung
:	aromatische Bindung
~	beliebige Anzahl an Bindungen
@	Beliebige Ringbindung

unique SMILES (dt. eindeutiger SMILES) erzeugt. Kanonisierte SMILES werden in dieser Arbeit eingeführt, da in Abschnitt 2.5 des Maschinellen Lernens unique SMILES benötigt werden. Weininger et al. [121] präsentierten 1989 einen zweistufigen Algorithmus, der aus einem SMILES einen unique SMILES erzeugt. Als erstes wird das Molekül in einen Graphen, der Knoten und Kanten aufweist, umgewandelt (siehe Abschnitt 2.2). Dabei wird jedes Atom kanonisch geordnet und bekommt ein Label, das eine Zahl aufweist. Anschließend wird mit dem Atom, das die geringste Labelzahl zugeordnet bekommen hat, begonnen, den eindeutigen Graphen aufzubauen. Eine formale Beschreibung zur Kanonisierung von SMILES und Erzeugung von unique SMILES erhält man in der Arbeit von Weininger et al. [121]. Sie verbessert das Ergebnis zur Klassifizierung der Daten erheblich, da sie eindeutig beschrieben werden. Tabelle 3 verdeutlicht die Unterschiede zwischen SMILES und unique SMILES. Unique SMILES werden bei den Algorithmen zur Substruktur nicht benötigt, da SMILES hier in Graphen umgewandelt werden und der Algorithmus auf diesen arbeitet.

SMILES sind immer gültige SMARTS-Ausdrücke, allerdings sind SMARTS Ausdrücke nicht immer gültige SMILES-Ausdrücke. SMILES beschreiben Moleküle, SMARTS hingegen Pattern (dt. Muster). Durch die erweiterte SMILES-Regelmenge ist es mittels der SMARTS-Notation möglich, Substrukturen zu definieren. SMARTS besitzen zusätzliche Konventionen gegenüber der SMILES-Notation, um Atom- und Bindungssymbole zu beschreiben. Die Substruktursuche in einer Datenbank von Molekülen ist ein komplexes und rechenintensives

Tabelle 3: Unique SMILES
Wie bereits beschrieben, ist die SMILES-Notation nicht eindeutig. Durch den Einsatz von unique SMILES existiert für ein Molekül nur eine eindeutige Schreibweise. Auf der linken Seite der Tabelle sind die unterschiedlichen Möglichkeiten aufgeführt, wie das gleiche Molekül beschrieben werden kann. Auf der rechten Seite die unique SMILES-Notation. Wie zu sehen ist, werden die Moleküle auf der linken Seite alle durch das gleiche unique SMILES beschrieben.

SMILES	**Unique SMILES**
OCC	CCO
[CH3][CH2][OH]	CCO
C-C-O	CCO
C(O)C	CCO

Problem (siehe Kapitel 7). Die Substruktursuche ist z. B. Kernbestandteil der Arzneimittelforschung.

Bedingt durch die kompakte String-Schreibweise – ein typischer SMILES benötigt 50% bis 70% weniger Speicherplatz als eine äquivalente tabellarische Schreibweise – gehen gegenüber der tabellarischen Schreibweise in der SMILES- und SMARTS-Notation einige Informationen verloren. Daher weisen SMILES und SMARTS auch einige Nachteile bei der Beschreibung chemischer Strukturen auf. Sie beschreiben Moleküle nicht eindeutig. Ein Molekül kann in SMILES und SMARTS auf unterschiedliche Weise beschrieben werden (siehe Abbildung 2).

Eine weitere Nomenklatur zum Beschreiben von chemischen Strukturen, die im Grunde eine Erweiterung der SMILES- und SMARTS-Notation ist, ist die SYBYL-Line-Notation (SLN). Im folgenden Abschnitt werden die groben Unterschiede und die Gemeinsamkeiten aufgezeigt [73].

2.1.4 *SYBYL-Line-Notation*

Der nachfolgende Abschnitt wurde, soweit nicht anders angegeben, aus dem Buch von Gasteiger entnommen [56].

Die SYBYL-Line-Notation ist ebenfalls eine Nomenklatur zum Beschreiben von molekularen Strukturen. Die SLN beschreibt Strukturen ähnlich zu den SMILES und SMARTS aus den Abschnitten 2.1.1 und 2.1.2. Der Hauptunterschied ist, dass in der SLN-Syntax alle Wasserstoffatome genau spezifiziert werden müssen, da Valenzelektronen

nicht festgelegt sind. Sie kann ebenso wie die SMARTS-Notation Substrukturen darstellen. Die SLN verwendet sechs grundlegende Definitionen zur Darstellung einer chemischen Struktur.

Ähnlich wie in SMARTS und SMILES werden die Atome mit ihren Elementsymbolen dargestellt. Doppelbindungen werden durch ein Gleichheitszeichen (=), Dreifachbindungen mit einem (#) und aromatische Bindungen durch einen Doppelpunkt (:) beschrieben. Branches werden wie in SMARTS und SMILES durch Klammern ( ) dargestellt. Anders als bei SMILES und SMARTS werden Wasserstoffatome (H-Atome) explizit geschrieben [56].

Ein Ringschluss wird ähnlich wie bei SMILES und SMARTS durch Zuordnung von Zahlen dargestellt. Die aufgebrochenen Zyklen erhalten an den Atomen, an dem der Zyklus geöffnet wird, eine Zahl, die in eckigen Klammern geschrieben wird. Der Ringschluss erhält zusätzlich zur Zahl, vorangestellt das @-Zeichen. Siehe folgendes Beispiel:

C[15]H2CH2CH2CH2CH2CH2@15

Der rote Teil ist der öffnende Teil, der grüne der schließende Teil einer Ringbindung.

Zusätzlich können ähnlich wie in der SMILES- und SMARTS-Notation Atome und Bindungen weitere Attribute aufweisen wie z. B. eine zusätzliche Beschreibung zur Ladung oder Valenz eines Atoms. Diese werden in eckige Klammern [ ] hinter einen SLN-String geschrieben [56].

Im Folgenden sind vier unterschiedliche Schreibweisen für das Molekül *Tyrosin* dargestellt. Die erste ist eine SMILES-, die zweite eine SMARTS-, die dritte ein unique SMILES und die vierte die SLN-Schreibweise [56]:

SMILES: `C1=C(C=CC(=C1)O)CC(N)C(=O)O`
SMARTS: `c1cc(ccc1-[#6]-[#6](-[#6](=[#8])-[#8])-[#7])-[#8]`
Unique SMILES: `C1=CC(=CC=C1CC(C(=O)O)N)O`
SLN: `OHC(=O)CH(NH2)CHC[1]=CHCH=C(OH)CH=CH@1`

Die Darstellung von chemischen Strukturen hat eine Entwicklungsstufe erreicht, die sich in der Grundstruktur in Zukunft nicht wesentlich ändern sollte. Moderne Linearnotationen wie z. B. die der SMILES-Notation werden auch in Zukunft zur Darstellung von chemischen Strukturen genutzt werden [56].

Chemische Strukturen, genauer Moleküle, werden in der Chemoinformatik zur Informationsverarbeitung in Form von Graphen dargestellt. Graphen sind für den Nutzer wesentlich intuitiver als String-Notationen. Um String-Notationen lesen zu können, wird ein hoher Einarbeitungsaufwand benötigt.

2.2 GRAPHENTHEORIE

Die Graphentheorie findet in unterschiedlichen Bereichen wie z. B. der Physik, Chemie, Biologie, medizinischen Chemie, Ökonomie und der Informatik Anwendung [63, 92, 114, 118]. Chemische Strukturen können gut durch Graphen dargestellt werden. Hierbei können Atome durch Knoten, Bindungen durch Kanten, Atomgruppen durch Subgraphen oder gesamte Moleküle als Graphen dargestellt werden.

Ein Graph wird durch seine Knoten und Kanten definiert. Die in dieser Arbeit genutzten Graphen sind Multigraphen, die ungerichtet, gewichtet und gelabelt sind. Knoten und Kanten weisen Knoten- und Kantenlabel auf. Schleifen werden nicht berücksichtigt. Schleifen sind Kanten, die in einem Knoten starten und im selben enden. Im Folgenden werden die in dieser Arbeit genutzten Definitionen formal eingeführt. Sie stammen aus den Büchern [38, 56, 58].

2.2.1 *Graphen*

Sei $G = (V, E)$ ein *Graph*, $V(G)$ die nichtleere endliche Menge an Knoten und $E(G)$ die endliche Menge an Kanten. Die Kanten stellen eine Menge von $E \subseteq VxV$ dar. Sind die Knotenpaare aus E geordnet, spricht man von einem gerichteten Graphen und die Elemente von E werden mit Bögen bezeichnet. Sind diese hingegen ungeordnet, ist der Graph ungerichtet und die Elemente von E werden Kanten genannt. Der Unterschied in der Darstellung zwischen einem *gerichteten* und einem *ungerichteten Graphen* ist, dass die Kanten hier nicht durch Linien sondern durch Pfeile $\rightarrow$ beschrieben werden. Dies soll verdeutlichen, dass Kanten nur in eine vorgegebene Richtung durchlaufen werden können, hier im Beispiel von Knoten u nach Knoten v ($u \xrightarrow{e} v$). Seien $n = n(G) = |V|$ und $m = m(G) = |E|$ die Anzahl an Knoten und Kanten im Graphen G.

Da Moleküle mehrere Verbindungen zwischen einzelnen Atomen aufweisen können, werden *Multigraphen* zum Beschreiben von Molekülen genutzt. Im *Multigraph* G können zwei Knoten durch mehrere Kanten miteinander verbunden sein. Um Kreise von einer Multikante zu unterscheiden, werden Multikanten in gewichtete Kanten umgewandelt. Besitzen zwei Knoten z. B. eine Doppelbindung zueinander, wird diese Doppelbindung in eine Kante mit einem Gewicht von zwei umgewandelt.

Ein *gewichteter Graph* G weist den Kanten $e \in E$ einen festen Wert zu. Eine Kantengewichtsfunktion ist wie folgt definiert: $w : E \to \mathbb{N}$.

Zwei Knoten sind *adjazent* (benachbart), falls sie mit einer Kante verbunden sind. Zwei Kanten sind *inzident*, falls sie einen gemeinsamen Knoten aufweisen. Der *Grad* eines Knotens ist die Anzahl an *inzidenten* Kanten, die den Knoten enthalten.

2.2.2 *Subgraphen*

Molekulare *Substrukturen* sind Subgraphen eines Graphen G, diese werden nachfolgend definiert. Ein Graph $G' = (V', E')$ ist ein *Subgraph* von $G = (V, E)$, falls $V' \subseteq V$ und $E' \subseteq E$, d. h. die Knoten- und Kantenmenge von G' Teilmengen der Knoten- und Kantenmengen von G sind.

Ein *Subgraph* G' heißt *induzierter Subgraph* des Graphen G, falls zwei Knoten aus $V(G')$ immer genau dann in G' adjazent sind, wenn sie in G adjazent sind.

Graphen in dieser Arbeit weisen ebenso Knoten und Kantenlabel auf. Knotenlabel werden genutzt, um sie z. B. mit chemischen Elementsymbolen zu belegen oder deren Substruktur zu beschreiben. Kantenlabel werden verwendet, um ein Knotenpaar eindeutig zu darzustellen, d. h. um sicherzustellen, dass alle Kanten die richtigen Endpunkte erhalten. Beispiel: Ein Knotenpaar `C-N` weist ein Kantenlabel CN auf. Das stellt sicher, dass ein C-Atom nur mit einem N-Atom verbunden wird.

2.2.3 *Gelabelte Graphen*

Sei G ein *knoten- und kantengelabelter Graph*, $\mathcal{L}_V^G$ eine Knoten- und $\mathcal{L}_E^G$ eine Kantenlabelmenge. Ein Graph G über $(\mathcal{L}_V^G, \mathcal{L}_E^G)$ ist ein Tupel $G = (V^G, E^G, N^G, L_V^G, L_E^G)$. Sei V^G die Knotenmenge, E^G die Kantenmenge, $N^G : E^G \to V^{G^2}$ die eindeutige Zuordnung von Kanten zu ihren Knoten, d. h. $L_V^G : V^G \to \mathcal{L}_V^G$ weist jedem Knoten und $L_E^G : E^G \to \mathcal{L}_E^G$ jeder Kante ein Label zu und sei $(V^G)^2$ eine Teilmenge der Kardinalität 2 von V^G.

2.2.4 *Graphkomponenten*

Im Folgenden werden weitere Begrifflichkeiten aus der Graphentheorie beschrieben. Ein *Weg* in einem Graphen ist eine Folge von Kanten, durch die zwei Knoten des Graphen miteinander verbunden sind. Einen Weg bezeichnet man als *Pfad*, falls alle Knoten in der Folge von Wegen unterschiedlich sind. Um chemische Ringstrukturen zu

beschreiben, werden in der Graphentheorie Zyklen genutzt. Ein *Zyklus* in einem Graphen G ist ein Weg, bei dem der Start- und Endknoten gleich sind. Ein zusammenhängender Graph ohne Zyklus und Mehrfachkanten heißt *Baum*. Ein *Wald* besteht aus mehreren Bäumen.

In einem Graphen G sind zwei Knoten $v_i, v_j \in V$ Teil einer *Zusammenhangkomponente*, falls im Graphen G ein Weg existiert, der die Knoten v_i und v_j miteinander verbindet.

2.2.5 *Graph-Isomorphismen*

Gibt es zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ eine bijektive Abbildung $\varphi : V_1 \to V_2$ mit $u, v \in E_1 \Leftrightarrow \varphi(u), \varphi(v) \in E_2$ auf, sind die Graphen G_1 und G_2 *isomorph* zueinander.

Seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ zwei Graphen. Eine injektive Abbildung $\varphi : V_1 \to V_2$ heißt Subgraph-Isomorphismus von G_1 auf G_2, falls für alle $u, v \in V_1$ folgendes gilt: $(u, v) \in E_1 \Rightarrow (\varphi(u), \varphi(v)) \in E_2$.

Seien $G_1 = (V_1, E_1, s_1, t_1)$ und $G_2 = (V_2, E_2, s_2, t_2)$ Graphen wie in Abschnitt 2.2.1. Die Abbildungen $s : E \to V$ und $t : E \to V$ weisen den Kanten ihre Anfangs- und Endknoten zu. Ein Graph-Homomorphismus φ bildet einen Graphen G auf einen Graphen H $\varphi : G \to H$ ab, so dass eine Aggregierung von Knoten nur genau dann erreicht werden kann, falls auch die Kanten kongruent zusammengefasst werden können. φ besteht aus zwei Funktionen $\varphi_V : V_1 \to V_2$ und $\varphi_E : E_1 \to E_2$, so dass folgendes gilt:

$$s_2 \circ \varphi_E = \varphi_V \circ s_1$$

und

$$t_2 \circ \varphi_E = \varphi_V \circ t_1.$$

Sei G ein Multigraph. Ein partieller Graph-Homomorphismus ist ein Graph-Homomorphismus eines Teilgraphen $S \subseteq G$ zu einem Graphen H [38, 56, 58].

2.3 GRAPHERSETZUNGSSYSTEME

In den folgenden Abschnitten werden unterschiedliche Graphersetzungssysteme rudimentär eingeführt und beschrieben. Für grundlegendere Beschreibungen sei auf das Buch von Rozenberg et al. [62] verwiesen. Die folgenden Abschnitte sind, soweit nicht anders angegeben, aus dem Buch von Rozenberg et al. [62] entnommen worden.

Ein Graphersetzungssystem definiert eine Sprache von Graphen. Es besteht aus einer Menge an Regeln, die definieren, wie ein Graph

in einen anderen umgewandelt werden kann. Die entsprechende Sprache ist die Menge aller Graphen, die aus den Regeln hergeleitet werden können [109]. Graph-Grammatiken bieten im Allgemeinen eine intuitive Beschreibung zur Manipulation von Graphen und grafischen Strukturen, wie sie in Semantiken von Programmiersprachen, Datenbanken und Betriebssystemen auftreten können, an.

Eine Graph-Grammatik besitzt im Allgemeinen eine endliche Menge an Produktionen. Eine Produktion ist ein Tupel (M, D, E), wobei M der Mutter-, D der Tochtergraph und E ein Einbettungsmechanismus ist. Eine Produktion kann immer dann auf einen (modifizierten) Graphen H, auch Hostgraph genannt, angewendet werden, falls M eine Teilmenge von H ist. Eine Produktion wird angewendet, indem M aus H entfernt und durch eine isomorphe Kopie von D ersetzt wird. Eine Produktion beschreibt hierbei, wie M entfernt und durch einen Graphen D ersetzt werden kann. D wird durch die in E festgelegten Regeln in den Restgraphen H' von H eingebettet.

Es existieren primär zwei Arten der Einbettung (engl. embedding), die Klebe- (engl. gluing embedding) und die Verbindeeinbettung (engl. connecting embedding). Die Klebeeinbettung wird auch algebraischer und die Verbindeeinbettung algorithmischer Ansatz genannt. Die Verfahren erhielten ihre Namen durch die mathematischen Techniken, die hierbei genutzt werden [62].

Im algorithmischen Ansatz weisen Graph-Grammatik-Produktionen die Form (M, D, E) auf, wobei M hier nur aus einem Knoten besteht. Diese Knoten werden durch Graphen ersetzt. Darüber hinaus wird verlangt, dass das Ergebnis einer Folge von Knotenersetzungen nicht von der Reihenfolge abhängen darf, d. h. für jede Sequenz an Knotenersetzungen muss das gleiche Ergebnis folgen. Dies ermöglicht es Graph-Grammatiken zu generieren, die rekursiv definierte Mengen an Graphen beschreiben können. Im algorithmischen Ansatz gibt es mehrere Möglichkeiten, den Einbettungsmechanismus E zu definieren, und es gibt mehrere Einschränkungen bezüglich der Form der rechten Seite D. Im algebraischen Ansatz werden statt Knoten Kanten oder auch Hyperkanten (siehe Abbildung 8) ersetzt.

Es wird in Abschnitt 2.3.1 mit der Beschreibung der Node-Replacement Graph-Grammars (dt. knotenersetzende Graph-Grammatik, NRG) begonnen. In Abschnitt 2.3.2 folgt die Beschreibung der Hyperedge Replacement Graph-Grammars (dt. Hyperkantenersetzende Graph-Grammatiken, HRG). Abschließend werden in Abschnitt 2.3.3 die algebraischen Push-Out-Ansätze vorgestellt.

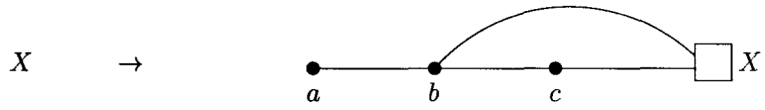


Abbildung 4: Produktionsregeln einer NLC-Grammatik
Die Abbildung zeigt eine Produktionsregel $X \to D$, die auf den Graphen H angewendet werden soll [62].

2.3.1 *Knotenersetzende Graph-Grammatiken*

In diesem Abschnitt werden NRGs eingeführt. Es wird beschrieben wie ein Graph anhand von Einbettungsprozessen im NRG-Ansatz in einen anderen Graphen überführt wird.

Ein Graph H wird in einen Graphen H' überführt, indem ein Subgraph M aus H entfernt und durch einen Graphen D ersetzt wird. Hierbei wird D in den Restgraphen von H eingebettet, d. h. H' ist der resultierende Graph, nachdem M aus H entfernt wurde.

Ein trivialer NRG-Ansatz ist der Node-Label-Controlled-Appoach (dt. Knotenlabel kontrollierter Ansatz, NLC). Produktionsregeln im NLC-Verfahren ersetzen gelabelte Knoten in einem ungerichteten Graphen und die Einbettungsverbindungsanweisungen verbinden den Tochtergraphen mit der Nachbarschaft des Mutterknotens. Sei eine NLC-Produktion $X \to D$ gegeben, wobei X ein Knotenlabel (nicht Terminalknoten) und D ein ungerichteter knotengelabelter Graph (mit terminalen und nicht terminalen Knoten) ist. In einem Graphersetzungssystem wird zwischen terminalen und nicht terminalen Knoten unterschieden. Dabei werden nur nichtterminale Knoten ersetzt, terminale Knoten hingegen nicht. Eine Produktionsregel kann auf jeden Knoten m im Hostgraphen H angewendet werden, der das Label X aufweist. Hierbei wird im Muttergraphen der Knoten m durch den Tochtergraphen D ersetzt. Alle Produktionen teilen sich die gleichen Verbindungsanweisungen. Eine Verbindungsanweisung hat die Form (μ, δ), wobei μ und δ (terminale und nicht terminale Knoten) Knotenlabel darstellen. Eine Verbindungsanweisung soll sicherstellen, dass im Einbettungsprozess jeweils eine Kante zwischen jedem mit δ gelabelten Knoten im Tochtergraphen D, mit jedem mit μ gelabelten Knoten, die Nachbarn des Mutterknotens m sind, erstellt wird. Falls m keinen mit μ gelabelten Nachbarn aufweist, kann die Verbindungsanweisung (μ, δ) nicht genutzt werden. Bei einer Verbindungsanweisung handelt es sich um ein geordnetes Paar. Eine endliche Anzahl an Verbindungsanweisungen wird auch als Verbindungsbeziehung bezeichnet.

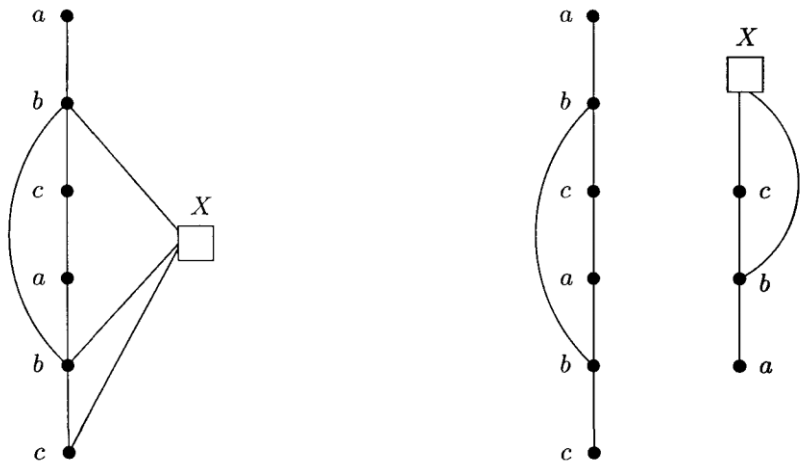


Abbildung 5: Ersetzungen in einer NLC-Grammatik
Die linke Abbildung zeigt den Hostgraphen H, die rechte die Ersetzung des Mutterknotens M durch den Tochtergraphen D [62].

Eine NLC Graph-Grammatik besteht aus einem Tupel $G = (\Sigma, \Delta, P, C, S)$. Seien $\Sigma - \Delta$ und Δ, (mit $\Delta \subseteq \Sigma$) Alphabete der Nichtterminal- und Terminalknotenlabel, P eine endliche Menge an Produktionen, C eine Verbindungsanweisung, d. h. eine binäre Relation über Σ und S der initiale Graph, auch das Startsymbol genannt (üblicherweise besteht dieser aus einem Knoten). Nichtterminale Knoten werden in Abbildung 5 durch Kästchen dargestellt. Die Sprache, die durch G generiert wird, ist $L(G) = \{H \in GR_\Delta | S \Rightarrow^* H\}$, wobei GR_Δ die Menge an ungerichteten Graphen darstellt, deren Knotenlabel in Δ enthalten sind. $\Rightarrow$ bezeichnet einen Ersetzungsschritt, hingegen $\Rightarrow^*$ eine Ableitung, d. h. eine Sequenz von Ersetzungsschritten.

Nachfolgend wird ein Beispiel einer NLC Graph-Grammatik illustriert. Sei $G = (\Sigma, \Delta, P, C, S)$ eine NLC Graph-Grammatik, so dass $L(G)$ die Menge aller Strings in $(abc)^+$ darstellt, die zusätzliche Kanten zwischen allen Knoten mit dem Label b aufweisen. Sei G wie folgt definiert: Sei $\Sigma = \{X, a, b, c\}$, $\Delta = \{a, b, c\}$, $S = X$ und P besteht aus den Produktionen wie in Abbildung 4 dargestellt (Produktion $X \to abc$ und $C = \{(c, a), (b, b), (b, X)\}$). Die durch diese Grammatik erzeugten Graphen sind Strings in $(abc)^*X$ mit zusätzlichen Kanten zwischen allen Knoten mit dem Label b oder X.

Seien (c, a), (b, b) und (b, X) eine Verbindungsanweisung, X ein nichtterminaler Knoten und a, b, c terminale Knoten. Sei auf der linken Seite der Abbildung 5 der Hostgraph H und sei m der Knoten von H mit dem Label X. Die Anwendung der Produktion $X \to D$ auf

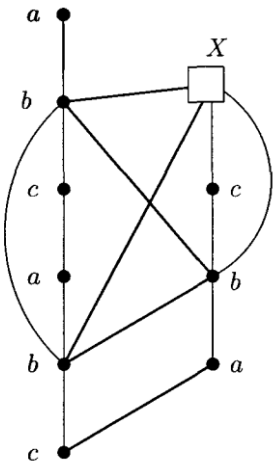


Abbildung 6: Resultierender Graph nach einer NLC-Ersetzung
Die Abbildung zeigt den resultierenden Graphen H' nach der Anwendung der Produktionsregel $X \to D$, auf den Graphen H [62].

den Tochterknoten m wird in zwei Schritten vollzogen. Der rechte Teil der Abbildung 5 zeigt, die Ersetzung von m durch D. Hierbei wird m und seine inzidenten Kanten entfernt und der Graph D zum Restgraphen H' von H hinzugefügt. In Abbildung 6 ist der transformierte Graph H' dargestellt. Die fünf in fett markierten Kanten sind diejenigen, die beim Verbindungsprozess hinzugefügt wurden. Nach Anwendung der Produktion $X \to D$ auf H ist H' der resultierende Graph [62].

2.3.2 *Hypergraphersetzende Graph-Grammatiken*

Der HRG-Ansatz wurde in den frühen Siebzigern von Feder [51] und Pavlidis [103] eingeführt und wird seit den späten siebziger Jahren intensiv untersucht [10, 29, 30, 45]. Hypergraphen sind verallgemeinerte Graphen, deren Hyperkanten eine beliebige Anzahl an Knoten aufweisen können (siehe Abbildung 7). Molekulare Strukturen können gut als Hypergraphen interpretiert werden, da Hyperkanten natürliche Beziehungen zwischen mehreren Molekülen darstellen können. Der HRG-Ansatz gehört zu den Klebeersetzungsansätzen [62].

Eine Hyperkante ist ein atomarer Gegenstand mit einer festen Anzahl an Tentakeln, auch $\mathtt{type}$ der Hyperkante genannt (siehe Abbildung 8). Sie kann an jede Struktur angehängt werden, die eine Menge an Knoten aufweist. Dabei wird jede Tentakel einer Hyperkante mit einem dieser Knoten verbunden. Hyperkanten steuern die Reihenfolge dieser Befestigungsknoten ($\mathtt{att}$-Knoten). Sie können auch die

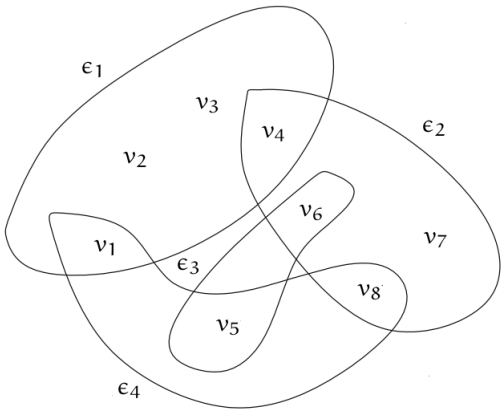


Abbildung 7: Hypergraph
Der obige Hypergraph hat 8 Knoten und 4 Hyperkanten [19].

Rolle eines Platzhalters einnehmen, dann durch eine andere Struktur ersetzt werden kann.

2.3.2.1 *Hypergraph-Ersetzungen*

Eine HRG ist ein elementarer Ansatz zum Umschreiben von Graphen und Hypergraphen. Eine HRG entfernt eine Hyperkante und ersetzt sie durch eine Struktur R, die in die Original Struktur eingebunden wird (siehe Abbildung 9). R wird anschließend mit dem Rest der ursprünglichen Struktur überdeckt und verbunden, indem jeder externe Knoten mit dem entsprechenden anzuhängenden Knoten verschmolzen wird. Ein externer Knoten ist ein Knoten, der maximal nur eine Verbindung zu einer Hyperkante aufweist.

Das Ersetzen einer Hyperkante durch eine Struktur R kann wiederholt werden, falls die ursprüngliche Struktur oder die zu ersetzende Struktur mit weiteren Hyperkanten versehen ist. Falls Hyperkanten mit Labeln markiert sind, können Produktionen definiert werden. Graphtransformationsansätze bestehen in der Regel aus Produktionen der Form

$$p : L \to R,$$

wobei L die linke Seite und R die rechte Seite einer Produktion darstellt. HRGs bestehen aus einer Startstruktur und einer endlichen Menge an Produktionen. Falls eine Hyperkante, die mit einem Label der linken Seite einer Produktion gekennzeichnet ist, durch die rechte Seite ersetzt wird, wird dies als eine direkte Ableitung bezeichnet. Der duale Graph D eines Hypergraphen H ist der Graph, der genau einen Knoten für jede Hyperkante aus H und eine Hyperkante für jeden Knoten aus H aufweist. Dazu muss die Anzahl an externen Knoten in R dem Typ der zu ersetzenden Hyperkante entsprechen.

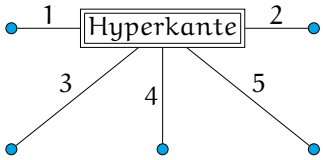


Abbildung 8: Hyperkante und deren Tentakeln
Die Anzahl an Tentakeln einer Hyperkante bestimmt den Typ einer Hyperkante. Die hier abgebildete Hyperkante ist vom Typ 5. Tentakeln sind Knoten, die mit der Hyperkante verbunden sind. Zur Unterscheidung der Tentakeln werden sie in diesem Beispiel mit natürlichen Zahlen nummeriert, die den Index der Tentakeln angibt. Jede Hyperkante kann eine beliebige Anzahl an Knoten aufweisen. Die Blattknoten sind Knoten mit nur einer Kante und in diesem Beispiel externe Knoten (blau markierte Knoten).

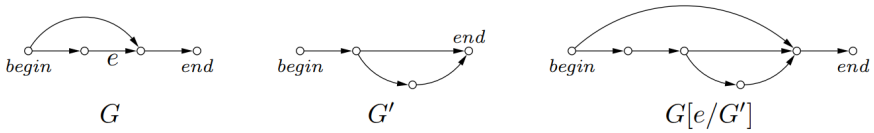


Abbildung 9: Ersetzung einer Kante
Die Abbildung zeigt den resultierenden Graphen G nach der Ersetzung einer Kante e durch einen Graphen G' [62].

Ein kurzes Beispiel wird nachfolgend vorgestellt. Sei hierbei ein gerichteter Multigraph G gegeben und weise dieser Graph zwei gelabelte Knoten $\mathtt{begin}$ und $\mathtt{end}$ auf. Eine Kante e kann hierbei aus G durch einen anderen Graphen G' ersetzt werden, indem e in G entfernt und G' zu G hinzugefügt wird. Hierbei wird der $\mathtt{begin}$-Knoten von G' mit dem Startknoten der Kante e und der $\mathtt{end}$-Knoten von G' mit dem Endknoten der Kante e verbunden (siehe Abbildung 9). Sei $G[e/G']$ der resultierende Graph. Dabei bleiben die Knoten $\mathtt{begin}$ und $\mathtt{end}$ von G erhalten.

Im Folgenden werden Kanten mit beliebigen Symbolen gelabelt. Dadurch ist es möglich, Produktionen zu definieren. Die linken Seiten stellen Label und die rechten Graphen dar. Eine Produktion $S ::= G'$ wird auf einen Graphen G angewendet, indem eine mit S gelabelte Kante mit G' ersetzt wird. Dies stellt eine direkte Ableitung $G \Rightarrow G[e/G']$ dar.

Die in 10 abgebildeten Produktionen generieren eine Menge an seriell-parallelen Graphen, falls mit einer S gelabelten Kante begonnen wird und so lange Produktionen abgeleitet werden, bis keine dieser S-Kanten mehr vorhanden sind. Seriell-parallele-Graphen mit

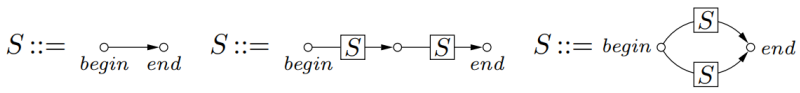


Abbildung 10: Produktionen einer HRG
Die Abbildung stellt Produktionsregeln einer HRG dar [62].

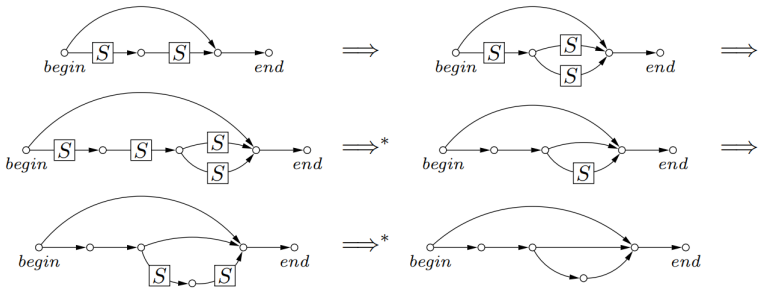


Abbildung 11: Produktionen einer HRG
Die Abbildung zeigt Produktionsregeln einer HRG zum Generieren eines seriell-parallelen Graphen [62].

zwei distinkten Knoten, die als Terminale und rekursiv durch zwei einfache Anordnungen gebildet werden können, bezeichnet. Sie werden in der Regel bei der Modellierung von Stromkreisen verwendet [49]. In Abbildung 11 ist eine beispielhafte Ableitung einer HRG dargestellt. $\Rightarrow^*$ bezeichnet dabei den transitiven und reflexiven Abschluss von $\Rightarrow$. Ersetzungen, die hier vorgestellt werden, nennen sich Kantenersetzungen.

2.3.3 *Algebraische Graphersetzungen*

Der algebraische Ansatz wurde in den frühen siebziger Jahren von Ehrig, Pfender und Schneider an der Technischen Universität Berlin entwickelt [47]. Der erste algebraische Ansatz, der vorgestellt wird, ist der DPO-Ansatz. Dieser wird DPO-Ansatz genannt, weil er auf zwei Pushout-Konstruktionen basiert. Im Gegensatz zum DPO-Ansatz definiert der SPO-Ansatz einen einzelnen Pushout durch eine direkte Ableitung. Die algebraischen Ansätze zur Graphtransformation basieren auf dem Konzept des Verkleben von Graphen. In den Abschnitten 2.3.3.1 und 2.3.3.2 werden beide Ansätze vorgestellt. Zuerst folgen informelle, später formelle Aussagen zu den Ansätzen. Der Abschnitt endet mit einem Vergleich zwischen beiden Ansätzen.

2.3.3.1 *Double-Pushout-Approach*

Bei der Spezifikation von Graphersetzungssystemen hat der algebraische Ansatz seit den 1960 Jahren sehr an Bedeutung gewonnen. Dieser wurde mit dem Ziel entwickelt, Grammatiken von Wörtern auf Graphen zu verallgemeinern. Dieser wird seit mehreren Jahren erforscht [64]. Der erste algebraische Ansatz zur Graphtransformation, der sogenannte DPO-Ansatz, der seinen Namen der grundlegenden algebraischen Konstruktion verdankt, wurde durch Volker et al. im Jahr 1979 eingeführt [24]. Dies erlaubt einen Ansatz zu formulieren, der durch zwei Klebekonstruktionen modelliert wird.

In den algebraischen Ansätzen wird ein Graph als zwei sortierte Algebren betrachtet, wobei die Knoten V und Kanten E als Träger von Informationen angesehen werden können, weil die Quelle $s : E \to V$ und die Senke $t : E \to V$ zwei unäre Operationen darstellen. Zusätzlich existieren Labelfunktionen $lv : V \to L_V$ und $le : E \to L_E$. L_V und L_E besitzen eine endliche Anzahl an Labeln. Kanten stellen eigenständige Objekte dar, daher können mehrere Kanten mit dem gleichen Label verwendet werden.

Jede Produktion definiert ein partielles Matching zwischen der rechten und linken Seite. Dabei wird bestimmt, welche Knoten und Kanten erhalten bleiben und welche entfernt werden, falls die Produktion p angewendet wird. Das Beispiel in Abbildung 12 zeigt auf, wie die Produktion $p_1 : L_1 \to R_1$ auf den Graphen G_1 angewendet wird. Die Produktion p_1 setzt drei Knoten auf der linken Seite L_1 und zwei Knoten auf der rechten Seite R_1 voraus. Die Knoten der rechten Seite sind zusätzlich mit einer Kante verbunden. Die Zahlen an den Knoten und Kanten definieren die partiellen Matchings zwischen der linken Seite L_1 und der rechten R_1. Zwei Elemente mit der gleichen Zahl sollen dasselbe Objekt vor und nach der Anwendung einer Produktion darstellen. Um p auf den Graphen G_1 anwenden zu können, muss das Vorhandensein von L_1 im Graphen G_1 geprüft werden, d. h. ein Match gefunden werden. Ein Match $m : L \to G$ für eine Produktion p ist ein Graphhomomorphismus, wobei Knoten und Kanten von L auf G gemappt werden, sodass die Struktur und Label erhalten bleiben. Das Match $m_1 : L_1 \to G_1$ der direkten Ableitung in Abbildung 12 mappt jedes Element von L_1 auf das Element von G_1, falls es die gleichen Zahlen aufweist. Falls die Produktion p_1 beim Matching von m_1 auf den Graphen G_1 angewendet wird, müssen alle Objekte in G_1, die Element von L_1 aber nicht von R_1 sind, gelöscht werden. In der Abbildung 12 ist das der Knoten 2, der gelöscht wird. Symmetrisch fügt man zu G_1 jedes Element von R_1 hinzu, das kein entsprechendes Element in L_1 ist, d. h. die Kante 5 wird in diesem Fall eingefügt. Alle übrigen Elemente von G_1 bleiben erhalten und der resultierende Graph H_1 ist durch Ableitung von $G_1 - (L_1 - R_1) \cup (R_1 - L_1)$ erzeugt worden.

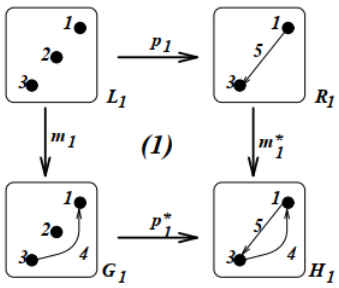


Abbildung 12: Produktion einer DPO
Die Abbildung zeigt eine direkte Ableitung im DPO-Ansatz [62].

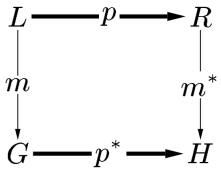


Abbildung 13: Schematische Abbildung einer direkten DPO Ableitung $G \overset{p,m}{\Longrightarrow} H$
Die Abbildung zeigt eine direkte Ableitung im DPO-Ansatz [62].

Die Anwendung einer Produktion kann als eine Einbettung in einen Kontext angesehen werden, der Teil eines gegebenen Graphen G, aber nicht Teil des Matchs ist. In diesem Beispiel ist das die Kante 4 aus Abbildung 12. Am Ende einer direkten Ableitung bezieht sich die Produktion $p_1^* : G_1 \to H_1$ auf den abgeleiteten Graphen und stellt sicher, dass alle Elemente durch die direkte Ableitung erhalten bleiben. Das Match m_1^* ordnet die rechte Seite der Produktion R_1 auf ihr Vorkommen im abgeleiteten Graph H_1 ab. Eine direkte Ableitung von G nach H, die bei Anwendung einer Produktion p bei Übereinstimmung von m entsteht, ist schematisch in Abbildung 13 dargestellt und wird mit $d = (G \overset{p,m}{\Longrightarrow} H)$ bezeichnet.

2.3.3.2 *Single-Pushout-Approach*

Im Jahr 1993 wurde von Kor ein zweiter algebraischer Ansatz präsentiert, der nur einen einzigen Pushout definiert [81]. Produktionen werden im SPO-Ansatz anders dargestellt als im DPO-Ansatz. Sie besitzen keinen Klebegraphen K mehr, sondern werden unter Zuhilfenahme eines partiellen Graph-Homomorphismus beschrieben. Anstatt zwei Ableitungen, wie im DPO-Ansatz zu verwenden, wird im SPO-Ansatz nur eine Ableitung angewandt.

In diesem Abschnitt erhält man eine kurze Einführung zum SPO-Ansatz zur Graphtransformation.

Definition 2.1 (Partieller Graphmorphismus)**.**
Sei $G = (G_V, G_E, s^G, t^G, lv^G, le^G)$ *ein Graph. Sei hierbei* G_V, G_E *die Menge an Knoten und Kanten von* G, s^G, t^G *die Start- und Zielkomponente eines Mappings und* lv^G, le^G *die Zuweisung der Knoten- und Kantenlabel. Sei* S *ein Subgraph von* G *mit* $S_V \subseteq G_V$, $S_E \subseteq G_E$, $s^S = s^G|_{S_E}$, $t^S = t^G|_{S_E}$, $lv^S = le^G|_{S_E}$ *und* $le^S = le^G|_{S_E}$. *Ein (partieller) Graphmorphismus* g *von* G *auf* H *ist ein Graphmorphismus aus einigen Subgraphen* $dom(g)$ *von* G *auf* H. *Sei* $dom(g)$ *die Domain von* g.

Graphen, die eine endliche Anzahl an Labeln und zugleich einen partiellen Morphismus aufweisen, werden als Klasse $Graph^P$ bezeichnet. In der Regel bestehen Produktionen $L \xrightarrow{p} R$ aus zwei Graphen L und R, der rechten und linken Seite und dem partiellen Morphismus p dazwischen. Der Morphismus p verdeutlicht indes, welche Elemente der linken Seite der rechten entsprechen. Es werden die Elemente gelöscht, die nicht durch den Morphismus beschrieben werden. Alle Objekte, die durch den Morphismus erhalten bleiben, bilden den Anwendungskontext. Wenn ein Objekt auf der rechten Seite der Produktion unter dem Morphismus kein Urbild aufweist, wird es hinzugefügt. Man beachte, dass dieser Anwendungskontext mit dem des Klebegraphen K aus den DPO-Produktionen konsistent ist.

Definition 2.2 (Produktion und Graph-Grammatik)**.**
Eine Produktion $p : (L \xrightarrow{r} R)$ *besteht aus einer Produktion* p *und einem injektiven partiellen Graphmorphismus* r, *die Morphismusproduktion genannt wird. Sei eine Graph-Grammatik ein Paar* $G = ((p : r)_{p \in P}, G_0)$, *wobei* $(p : r)_{p \in P}$ *eine Familie an Morphismenproduktionen, die durch ihren Index beschrieben werden, dargestellt. Hierbei stellt* G_0 *den Startgraphen einer Grammatik dar.*

Um Produktionen bei direkten Ableitungen unterscheiden zu können, sollten sie eindeutig sein, d. h. zwei Produktionen sollten nicht den gleichen Index aufweisen. Es wird lediglich $p : L \to R$ als Notation verwendet, um eine Produktion $p : L \xrightarrow{p} R$ zu beschreiben. Hierbei stellt die Produktion p auch gleichzeitig den Namen der Produktion dar.

Abbildung 14 zeigt eine Produktion r, die eine Schleife einfügt und die Knoten 2 und 3 löscht. Der Anwendungskontext besteht aus dem Knoten 1 (durch die gestrichelte Linie gekennzeichnet). r definiert hierbei den Knoten 1, der ihn auf den einzigen Knoten auf der rechten Seite mappt. r ist hingegen nicht für die Knoten 2 und 3 definiert. Die Zahlen an den Knoten verdeutlichen den Morphismus, so dass die gestrichelte Linie später weggelassen werden kann. Der Graph L kann formal wie folgt beschrieben werden: Sei $L = (\{\cdot_1, \cdot_2, \cdot_3\}, \emptyset, s^L, t^L, lv^L, le^L)$, wobei s^L, t^L und le^L leere Funktionen darstellen. lv^L ist durch $v^L(\cdot_i) = *$ für $i = 1, 2, 3$ definiert.

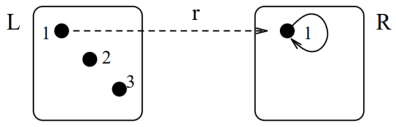


Abbildung 14: Eine SPO-Produktion r
Die Abbildung zeigt eine Produktion r im SPO-Ansatz, die zwei Knoten löscht und eine Schleife einfügt [62].

Nachfolgend werden die Gemeinsamkeiten und Unterschiede zwischen den vorgestellten algebraischen Ansätzen aus den Abschnitten 2.3.3.1 und 2.3.3.2 erläutert.

Zu den Grundeigenschaften des SPO-Ansatzes gehört die Vollständigkeit ihrer direkten Ableitungen, d. h. existiert eine Produktion, die angewendet werden kann, besteht auch immer eine entsprechende direkte Ableitung. Direkte Ableitungen sind im DPO-Ansatz aufgrund ihrer Verklebeeingenschaften nicht vollständig. Sie weisen aber direkte Ableitungen auf, die invertierbar sind. Dies ist beim SPO-Ansatz nicht der Fall. Die vorgestellten Ansätze weisen drei grundlegende Unterschiede bei der Beschreibung von Graphersetzungen auf. Diese sind:

1. die unterschiedliche Art einen Graphen zu beschreiben,
2. die Voraussetzungen für die Anwendung einer Produktion und
3. die Herangehensweise einen Graphen nach Anwendung einer Produktion zu erzeugen.

Diese Unterschiede definieren eine direkte Ableitung. Die DPO- und SPO-Ansätze nutzen die gleichen Typen an Graphen und auch die Produktionen in beiden Ansätzen sind ähnlich, werden aber unterschiedlich dargestellt. Eine Produktion im DPO-Ansatz $L \xleftarrow{l} K \xrightarrow{r} R$ ist ein injektiver Graphmorphismus, hingegen stellt eine SPO-Produktion einen partiellen Graphmorphismus $L \xrightarrow{p} R$ dar [109].

2.4 KOMPLEXITÄTSTHEORIE

Dieser folgende Abschnitt ist, soweit nicht anders angegeben, an das Buch von Kleinberg und Tardos angelehnt [80].

Die Komplexitätstheorie ist ein zentrales Fachgebiet der theoretischen Informatik. Unter Zuhilfenahme der Komplexitätstheorie können Berechnungsprobleme in Komplexitätsklassen eingeteilt werden. Die Komplexität einer Klasse gibt an, wie viel Ressourcen, wie z. B. Speicher oder Rechenzeit benötigt wird, um bestimmte Probleme zu

lösen. Die Algorithmik untersucht bestimmte Probleme sehr genau, die Komplexitätstheorie hingegen hat eine allgemeinere Sicht auf ganze Klassen von Problemen, die ähnliche Eigenschaften aufweisen. Hier möchte man wissen, ob ein betrachtetes Problem in einer bestimmten Komplexitätsklasse liegt oder nicht. Zur Analyse von Algorithmen wird die Rechenzeit in Abhängigkeit von Parametern wie z. B. der Eingabegröße gemessen. Das am häufigsten genutzte Rechenmaß zur Abschätzung des Aufwandes ist die maximale Rechenzeit (vereinfacht Laufzeit). Oft wird hier auch der Begriff worst-case Laufzeiten genutzt. Laufzeiten stellen dann Funktionen $f : \mathbb{N} \to \mathbb{N}$ dar. Aus diesem Grund werden sie meist nach ihrem Aufwand nach oben oder unten asymptotisch abgeschätzt. Die $\mathcal{O}$-Notation hat sich seit der ersten Verwendung durch Bachmann im Jahre 1892 durchgesetzt, um Wachstumsgeschwindigkeiten von Funktionen $t : \mathbb{N} \to \mathbb{R}^+$ zu beschreiben. Das Ziel ist es hierbei, die Beziehung wie z. B. $\leqslant$, $\geqslant$, $=$, $<$ und $>$ zwischen Funktionen zu charakterisieren [119]. Die Laufzeit eines Algorithmus liegt in $\mathcal{O}(f)$, falls eine Funktion $f : \mathbb{N} \to \mathbb{N}$ existiert, so dass die maximale Rechenzeit höchstens $c \cdot f(n)$, für eine Konstante $c \in \mathbb{N}$, ist. Diese Definition ist gegenüber Änderungen des Rechnermodells sehr robust.

Algorithmische Probleme können mehrere korrekte Antworten aufweisen. Man ist in der Regel mit der Ausgabe einer korrekten Antwort zufrieden. Sollte ein Problem mehrere korrekte Lösungen aufweisen, kann das Aufzählen dieser Lösungen sehr aufwendig sein. Bei der Berechnung eines kürzesten Weges wird die Lösung mit den geringsten Kosten gesucht. Diese Art der Probleme wird als Optimierungsproblem bezeichnet. Entscheidungsprobleme sind Probleme, die bei einer Eingabe w nur $\mathtt{ja}$ oder $\mathtt{nein}$ als Antwort ausgeben. Ein Entscheidungsproblem fragt, ob eine vorgegebene Bedingung erfüllbar ist.

Die Komplexitätsklasse $\mathcal{P}$ enthält alle Entscheidungsprobleme, die durch eine deterministische Turingmaschine in polynomieller Zeit gelöst werden können. Ein Problem A heißt hierbei effizient berechenbar, falls $A \in \mathcal{P}$ liegt. Algorithmen werden im Allgemeinen als effizient bezeichnet, falls die Anzahl an elementaren Operationen, die z. B. von einer Eingabelänge n abhängen, durch ein Polynom in n beschränkt werden können. Die Komplexitätsklasse $\mathcal{NP}$ ist die Klasse an Problemen, die sich nichtdeterministisch in Polynomialzeit verifizieren lassen. Der Name $\mathcal{NP}$ steht für nichtdeterministisch polynomiell. Die $\mathcal{NP}$-Vollständigkeit ist für Entscheidungsprobleme definiert, während man bei anderen Problemtypen, wie z. B. bei Optimierungsproblemen, von $\mathcal{NP}$-Äquivalenz spricht. Die zentrale Fragestellung der Komplexitätstheorie ist, ob $\mathcal{NP} \neq \mathcal{P}$ gilt. Es wird vermutet, dass $\mathcal{NP} \neq \mathcal{P}$ gilt und somit, dass die Probleme, die in $\mathcal{NP}$ liegen, nicht effizient berechnet werden können. Beim SAT-Problem, auch Erfüll-

barkeitsproblem der Aussagenlogik genannt, wird z. B. eine erfüllende Belegung für eine aussagenlogische Formel Φ gesucht. Das SAT-Problem gehört zur Klasse der $\mathcal{NP}$-Probleme, die zur Zeit nicht in polynomieller Zeit gelöst werden können. Das Erfüllbarkeitsproblem der Aussagenlogik ist $\mathcal{NP}$-vollständig.

2.4.1 $\mathcal{NP}$-*Vollständigkeit*

Das $\mathcal{NP} \neq \mathcal{P}$ Problem wurde vom Clay Mathematical Institute auf die Liste der sieben wichtigsten mit der Mathematik verbundenen Probleme aufgenommen. Für die Lösung des Problems wurde ein Preisgeld von 1.000.000 $ ausgesetzt [75].

Ein Problem L ist $\mathcal{NP}$-vollständig, falls es zur Klasse $\mathcal{NP}$ gehört und $\mathcal{NP}$-schwer ist. Genauer, ein Entscheidungsproblem ist $\mathcal{NP}$-vollständig, falls es in der Komplexitätsklasse $\mathcal{NP}$ liegt, d. h. ein polynomieller Verifizierer existiert, der in polynomieller Zeit entscheidet, ob eine gegebene Lösung eines zugehörigen Problems eine Lösung darstellt, mit der zusätzlichen Bedingung, dass das Entscheidungsproblem zu den $\mathcal{NP}$-schweren Problemen gehört. Die Klasse aller $\mathcal{NP}$-vollständigen Probleme wird mit $\mathcal{NPC}$ bezeichnet.

Definition 2.3 (Klasse $\mathcal{NP}$)**.**
Die Menge $\mathcal{NP}$ ist die Menge aller Entscheidungsprobleme X, die effizient in polynomieller Zeit überprüft werden können, so dass ein effizienter Algorithmus A mit zwei Eingaben s und t und ein Polynom p mit folgenden Eigenschaften existieren:

1. *Ist $s \notin X$, so ist $A(s,t) =$ nein $\forall t$.*
2. *Ist $s \in X$, $\exists t$ mit $|t| \leqslant p(|s|)$ und $A(s,t) =$ ja.*

Es existieren viele Probleme, mit denen man aufweisen kann, dass sie komplexitätstheoretisch vergleichbar sind [80]. Möchte man zeigen, dass ein Problem X mindestens so schwer ist wie ein Problem Y, kann dies unter Zuhilfenahme einer polynomiellen Reduktion erfolgen. Dabei wird ein Problem Y in polynomieller Zeit auf das Problem X reduziert (Kurzschreibweise $Y \leqslant_p X$).

Definition 2.4 (Polynomialzeitreduktion)**.**
Seien A und B zwei Entscheidungsprobleme mit $A, B \subseteq \Sigma^*$. A ist polynomiell auf B reduzierbar (geschrieben $A \leqslant_p B$), falls eine polynomiell berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ existiert, so dass für alle Wörter $w \in \Sigma^*$ die Äquivalenz $w \in A \Leftrightarrow f(w) \in B$ folgt [27].

Definition 2.5 ($\mathcal{NP}$-Vollständigkeit)**.**
Ein Problem X ist $\mathcal{NP}$-vollständig, falls $X \in \mathcal{NP}$ und $\forall\, Y \in \mathcal{NP}$, $Y \leqslant_p X$, d. h. dass jedes Problem in $\mathcal{NP}$ auf X in polynomieller Zeit reduziert werden kann.

Definition 2.6 (Cliquenproblem)**.**
Gegeben sei ein ungerichteter Graph $G = (V, E)$ *ohne Mehrfachkanten und* $U \subseteq V$*. Eine Clique ist eine Teilmenge* U *von* V*, die einen vollständigen Teilgraphen induziert. Ist* U *eine Clique, so gilt für einen Teilgraphen* $H = (U, F)$*, wobei* F *alle Kanten aus* E *enthält, die zwei Knoten in* U *verbinden, dass je zwei beliebige Knoten* v *und* w *aus* U *durch eine Kante miteinander verbunden sind.*

2.4.2 *Parametrisierte Algorithmen*

Die parametrisierte Algorithmik ist ein Teilgebiet der theoretischen Informatik und wurde von Rod Downey und Michael Fellows in den 1990er Jahren entwickelt [3, 40–43]. Die Hauptleistung ihrer Arbeit war es, eine umfassende Komplexitätstheorie für parametrisierte Probleme zu entwerfen.

Parametrisierte Algorithmen werden seit vielen Jahren untersucht, sowohl von der theoretischen, als auch von der praktischen Seite [20, 32, 37]. Sie sind ein Hilfsmittel, um die Grenze zwischen effizient berechenbaren und nicht effizient berechenbaren Problemen genauer zu beschreiben. Ziel ist es, die Schwierigkeit eines Problems nicht nur anhand der Größe der Eingabe zu beschreiben, sondern Einsichten in die Struktur der Eingabe zu gewinnen, die die Ursache der Schwierigkeit charakterisiert. Somit kann ein weiterer Parameter angegeben werden. So können oft auch große Eingaben gelöst werden, wenn der Strukturparameter k hinreichend klein ist. Die Kunst ist es, die Strukturen zu finden, die das Problem schwer machen.

Klassisch beschreibt man eine Eingaben in ihrer Eingabegröße n und sagt, dass ein Problem effizient lösbar ist, wenn es von einem Algorithmus gelöst werden kann, dessen Laufzeit durch ein Polynom in n beschränkt ist. Bei der parametrisierten Komplexität beschreibt man ein Problem in der Eingabegröße n und einem zusätzlichen Parameter k und sagt, dass ein parametrisiertes Problem fixed-parameter-tractable (FPT) ist, falls es von einem Algorithmus gelöst werden kann, dessen Laufzeit durch $f(k) \cdot p(n)$ beschränkt ist, wobei f eine berechenbare Funktion, k der Parameter, p ein Polynom und n die Eingabelänge darstellt. Eine Funktion heißt berechenbar, falls es einen Algorithmus gibt, der die Funktion berechnet. Die Komplexitätsklasse wird mit FPT bezeichnet. Das Ziel der parametrisierten Algorithmik besteht darin, Algorithmen zu entwerfen, wobei der Faktor $f(k)$ im Rahmen der Laufzeit so klein wie möglich gehalten werden soll [32]. FPT-Algorithmen sind meist für Eingaben bis $n = 100$ praktikabel. Brute-Force Algorithmen, die eine Laufzeit von $\mathcal{O}(2^n)$ aufweisen, sind bereits bei einer Eingabe von $n = 20$ nicht mehr effizient nutzbar [32, 44].

Probleme mit unterschiedlichen Parametern können sowohl in FPT als auch nicht in FPT sein, wie z. B. beim Cliquenproblem (siehe Definition 2.6). Falls der Parameter des Cliquenproblems die maximale Clique ist, ist es kein FPT-Algorithmus. Falls der Parameter der Maximalgrad des Eingabegraphen ist, hingegen schon. Man sagt auch, dass ein Problem parametrisierbar in dem entsprechenden Parameter ist. Daher hängt die Klassifizierung des Problems entscheidend von der Wahl des Parameters ab. Bei Optimierungsproblemen erkennt man sofort einen relevanten Parameter, nämlich den der in Größe der gesuchten Lösung.

Definition 2.7 (Parametrisiertes Vertex-Cover-Problem)**.**
Das Vertex-Cover-Problem fragt, ob es in einem gegebenen ungerichteten Graphen $G = (V, E)$ *und zu einer natürlichen Zahl* k *eine Knotenüberdeckung der Größe* $\leqslant k$ *existiert, d. h. existiert eine aus maximal* k *Knoten bestehende Teilmenge* $S \subseteq V$*, so dass jede Kante* $uv \in E$ *des Graphen* G *mit mindestens einem Knoten aus* S *verbunden ist, d. h. für jede Kante* $uv \in E$ *entweder* $u \in S$ *oder* $v \in S$ *oder beide sind in* S*.*

Karp zeigte, dass das Vertex-Cover-Problem ein $\mathcal{NP}$-vollständiges Problem ist [78]. Möchte man das Vertex-Cover-Problem mittels Brute-Force lösen, müsste man alle $\binom{n}{k}$ möglichen Teilmengen berechnen. Bei einer Eingabe von $n = 100$, $n = |V|$ und einem $k = 10$ existieren $\binom{100}{10}$= 17.310.309.456.440 Möglichkeiten, die es zu berechnen gilt. Wäre $k = 2$ würden sich die Möglichkeiten dementsprechend auf 4.950 verringern.

Eine FPT-Reduktion ist eine Funktion, die selbst in FPT-Zeit berechenbar ist und eine Instanz eines Problems P_1 auf eine Instanz eines Problems P_2 so abbildet, sodass $k_2 \leqslant g(k_1)$ für eine berechenbare Funktion g ist. Im Gegensatz zu FPT-Reduktionen kann sich bei einer Polynomialzeitreduktion die Größe der Lösung deutlich verändern. Aufgrund dieser Tatsache liefert die FPT-Reduktion eine feinere Abstufung als die Komplexitätsklasse $\mathcal{NP}$ und ermöglicht somit genauere Aussagen über die Komplexität von Problemen.

Definition 2.8 (Parametrisierte Reduktion)**.**
Seien $A, B \subseteq \Sigma^* \times \mathbb{N}$ *zwei parametrisierte Probleme. Eine parametrisierte Reduktion von* A *nach* B *ist ein Algorithmus, der bei einer Instanz* (x, k) *von* A *eine Instanz* (x', k') *von* B*, dann und nur genau dann ja ausgibt, falls:*

- (x, k) *ist eine Ja-Instanz von* A *ist und* (x', k') *eine Ja-Instanz von* B*,*
- $k' \leqslant g(k)$ *für eine berechenbare Funktion* g *und*
- *die Laufzeit* $f(k) \cdot |x|^{\mathcal{O}(1)}$ *für eine berechenbare Funktion* f *ist.*

Theorem 1.
Falls es eine parametrisierte Reduktion von A *nach* B *gibt und* B *ist in FPT, dann befindet sich* A *ebenso in der Klasse FPT.*

Es sei darauf hingewiesen, dass davon ausgegangen werden kann, dass die Funktionen f und g monoton wachsende Funktionen sind. Die berechenbare Funktion $g(k)$ kann durch $\hat{g}(k) = \max_{i=1}^{k} g(k) \geqslant g(k)$ ersetzt werden, was eine berechenbare monoton wachsende Funktion darstellt.

Beweis. Sei (x, k) eine Instanz von Problem A. Die parametrisierte Reduktion erzeugt eine äquivalente Instanz (x', k') in $f(k)|x|^{c_1}$ Zeit mit $k' \leqslant g(k)$ und $|x'| \leqslant f(k)|x|^{c_1}$. Angenommen, Algorithmus B weist eine Laufzeit von $h(k)n^{c_2}$ auf, wobei h eine nicht abnehmende Funktion ist. Durch Ausführung des Algorithmus auf (x', k') wird bestimmt, ob (x', k') eine Ja-Instanz von B und ob (x, k) eine äquivalente Ja-Instanz von A, mit einer Laufzeit von höchstens $h(k')|x'|^{c_2} \leqslant h(g(k))(f(k)|x|^{c_1})^{c_2}$, ist. Zusammen mit der Laufzeit der parametrisierten Reduktion ergibt sich eine gesamte Laufzeit von maximal $f'(k)|x|^{c_1 c_2}$, wobei $f'(k) = f(k) + h(g(k))(f(k))^{c_2}$ eine berechenbare Funktion ist. Somit ist B in FPT.

□

Theorem 2.
Falls es eine parametrisierte Reduktion von A *nach* B *gibt und von* B *nach* C*, dann existiert auch eine FPT-Reduktion von* A *nach* C*.*

Beweis. Sei $\mathcal{R}_1$ eine parametrisierte Reduktion von A nach B mit einer Laufzeit von $f_1(k) \cdot |x|^{c_1}$ und beschränktem Parameter $g_1(k)$ und sei $\mathcal{R}_2$ eine parametrisierte Reduktion von B nach C mit einer Laufzeit $f_2(k) \cdot |x|^{c_2}$ und beschränktem Parameter $g_2(k)$. Wie bereits erläutert sind die Funktionen f_1, f_2, g_1 und g_2 monoton wachsende Funktionen. Sei eine Instanz (x, k) von A gegeben. Reduktion $\mathcal{R}$ nutzt zur Erzeugung einer äquivalenten Instanz von B mit $|x'| \leqslant f_1(k) \cdot |x|^{c_1}$ und $k' \leqslant g_1(k)$ Reduktion $\mathcal{R}_1$ und dann Reduktion $\mathcal{R}_2$ auf (x', k'), um eine äquivalente Instanz von C mit $k'' \leqslant g_2(k')$ zu erzeugen. Eine Instanz (x, k) ist hierbei eine Ja-Instanz von A genau dann, wenn (x'', k'') eine Ja-Instanz von C ist. Hier ist $k'' \leqslant g_2(k') \leqslant g_2(g_1(k))$, unter Verwendung der Tatsache, dass g_2 eine nicht abnehmende Funktion ist, somit ist k'' beschränkt durch eine berechenbare Funktion von k. Die Laufzeit der Reduktion ist somit:

$$\begin{aligned} f_1(k)|x|^{c_1} + f_2(k')|x'|^{c_2} &\leqslant f_1(k)|x|^{c_1} + f_2(g_1(k)) \cdot (f_1(k) \cdot |x|^{c_1})^{c_2} \\ &\leqslant (f_1(k) + f_2(g_1(k))f_1(k)) \cdot |x|^{c_1 c_2} \\ &= f^*(k) \cdot |x|^{c_1 c_2}, \end{aligned}$$

wobei $f^*(k) = f_1(k) + f_2(g_1(k))f_1(k)$ eine berechenbare Funktion darstellt. In der ersten Ungleichung wurde ebenfalls die Tatsache genutzt, dass f_2 eine monoton wachsende Funktion ist. Diese Reduktion $\mathcal{R}$ erfüllt alle Bedingungen in Definition 2.8. □

2.4.2.1 *W-Hierarchie*

Downey und Fellows führten die $\mathcal{W}$-Hierarchie ein, um die genaue Komplexität verschieden harter parametrisierter Probleme zu erfassen. In diesem Abschnitt erhält man einen Einblick über die grundlegenden Definitionen und Ergebnisse der $\mathcal{W}$-Hierarchie. Ein Problem kann mit unterschiedlichen Parametern sowohl in FPT als auch nicht in FPT sein.

Definition 2.9 ($\mathcal{W}$-Hierarchie)**.**
Für $t \geqslant 1$ gehört ein parametrisiertes Problem P zur Klasse $\mathcal{W}[t]$, falls für ein $d \geqslant 1$ eine parametrisierte Reduktion von P nach $\mathcal{WCS}[C_{t,d}]$ existiert.

Das Cliquen-Problem z. B. ist $\mathcal{W}[1]$-vollständig. Daher wird es keinen FPT-Algorithmus für dieses Problem geben, solange $\mathcal{NP} \neq \mathcal{P}$ gilt, d. h. das Problem ist nicht FPT und FPT $\neq \mathcal{W}[1]$.

Im Weighted Circuit Satisfiability Problem (dt. Erfüllbarkeitsproblem für Schaltkreise, $\mathcal{WCS}$) ist eine boolesche Schaltung C und eine ganze Zahl k gegeben. Die Aufgabe besteht darin, zu entscheiden, ob C eine erfüllende Belegung mit einem Gewicht von genau k aufweist. Man überprüft alle $\mathcal{O}(n^k)$ möglichen Zuordnungen mit einem Gewicht von genau k und testet dann, ob es für C eine erfüllende Belegung gibt. Das Überprüfen kann für jedes feste k in polynomieller Zeit gelöst werden. Das Berechnen einer erfüllenden Belegung für C ist ein offensichtlich $\mathcal{NP}$-vollständiges Problem.

Definition 2.10 (Boolesche Schaltung und Weft)**.**
Eine boolesche Schaltung ist ein gerichteter azyklischer Graph, in dem die Knoten wie folgt markiert sind:

1. *Jeder Knoten mit einem Eingangsgrad von 0 ist ein Eingangsknoten,*
2. *jeder Knoten mit einem Eingangsgrad = 1 ist ein Negationsknoten,*
3. *jeder Knoten mit einem Eingangsgrad $\geqslant 2$ ist entweder ein Und- oder ein Oder-Knoten.*

Der Ausgangsknoten einer booleschen Schaltung weist einen Ausgangsgrad von 0 auf. Die Tiefe d eines $\mathcal{WCS}$ ist die maximale Länge auf einem Pfad zwischen dem Eingangs- und Ausgangsknoten. Die Weft t eines $\mathcal{WCS}$ ist die maximale Anzahl an Knoten mit einem Eingangsgrad > 2 auf einem Pfad vom einem Eingabe- zum Ausgabeknoten.

Am Anfang wird jedem Eingangsknoten ein Wert von 0 oder 1 zugewiesen. Die boolesche Schaltung weist eine erfüllende Belegung auf, falls der Wert des Ausgangsknotens gleich 1 ist. Es muss zwischen kleinen und großen Knoten unterschieden werden. Ein kleiner Knoten weist maximal einen Eingangsgrad kleiner gleich 2, große hingegen einen Eingangsgrad von größer 2 auf. Ein parametrisiertes Problem liegt in der Klasse $\mathcal{W}[i]$, falls jede Instanz (x, k) in eine boolesche Schaltung in FPT-Zeit transformiert werden kann und maximal eine Weftgröße von $i \in \mathbb{N}$ aufweist. Sei $C_{t,d}$ die Klasse an Schaltungen mit einer Weft von maximal t und Tiefe von höchstens d.

Definition 2.11 (Independent-Set)**.**
Sei $G = (V, E)$ *ein ungerichteter Graph ohne Mehrfachkanten. Gilt für je zwei beliebige Knoten* u *und* v *aus* U*, dass sie nicht benachbart sind, so nennt man* U *ein Independent-Set des Graphen* G*.*

Definition 2.12 (Dominating-Set)**.**
Sei ein Dominating-Set für einen Graphen $G = (V, E)$ *mit* $D \subseteq V$ *wie folgt definiert: Jeder Knoten* v*, der sich nicht in* D *befindet, ist mit mindestens einem Knoten aus* D *adjazent.*

Abbildung 15 zeigt am Beispiel des Independent- und des Dominatings-Set einen booleschen Schaltkreis $\mathcal{WCS}$.

Ein parametrisiertes Problem ist in XP (dt. Scheibenweise-Polynom-Algorithmen), falls es einen Algorithmus für das Problem gibt, dessen Laufzeit für jeden festen Wert des Parameters durch ein Polynom beschränkt werden kann. XP-Algorithmen weisen im Gegensatz zu FPT-Algorithmen eine weniger effiziente Laufzeit der Form $f(k) \cdot n^{g(k)}$ auf, wobei $g(k)$ eine berechenbare Funktion darstellt.

2.5 MASCHINELLES LERNEN

Die Abschnitte zum maschinellen Lernen wurden, soweit nicht anders angegeben, aus Da Silva et al. [33] und Friedman et al. [54] entnommen.

Die erste Neurocomputing-Publikation [93] stammt aus dem Jahr 1943, als McCulloch und Pitts das erste mathematische Modell entwarfen. Inspiriert wurden sie durch die biologischen Neuronen im menschlichen Gehirn. Es wird geschätzt, dass das biologische neuronale Netzwerk eines Menschen aus etwa 10^{11} Neuronen besteht. Jedes Neuron ist durchschnittlich mit weiteren 6.000 Neuronen verbunden, wodurch insgesamt 600 Billionen Synapsen im neuronalen Netzwerk existieren [116].

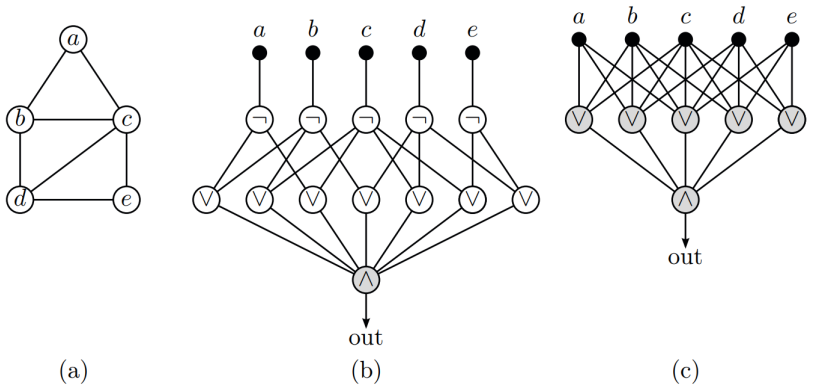


Abbildung 15: WCS-Schaltung
(a) bildet einen Graphen G mit fünf Knoten und sieben Kanten ab. (b) zeigt eine boolesche Schaltung mit einer erfüllenden Belegung für das Independent-Set für G mit einer Tiefe von 3 und einer Weft von 1. (c) zeigt eine boolesche Schaltung mit einer erfüllenden Belegung für das Dominating-Set für G mit einer Tiefe von 2 und einer Weft von 2 auf, jeder Oder-Knoten entspricht in (c) einem Knoten aus G. Jeder Oder-Knoten ist mit sich selbst und seinen Nachbarn aus G verbunden [32].

Künstliche neuronale Netze (KNN) verfügen über die Fähigkeit, Wissen zu abstrahieren. Sie können als eine Menge von Verarbeitungseinheiten definiert werden, die untereinander durch Mehrfachverbindungen (künstliche Synapsen) miteinander verbunden sind. Sie versuchen das menschliche Gehirn zu adaptieren und stellen vereinfachte Modelle biologischer Neuronen dar [69]. KNNs stellen Lernalgorithmen dar, die auf Basis einer gegebenen Lernmenge versuchen Daten zu generalisieren. Es existieren unterschiedliche KNN-Lernverfahren. In Abschnitt 2.5.1 werden Grundlagen zu den künstlichen neuronalen Netzen erläutert. In Abschnitt 2.5.2 werden rekurrente neuronale Netzwerke (RNN) eingeführt. RNNs werden insbesondere bei der Sprach- und Handschrifterkennung eingesetzt [60, 61, 97]. In Kapitel 9 werden sie zur Klassifikation von Molekülen eingesetzt. Zusätzlich zu den RNNs existieren Convolutional Neural Networks (dt. faltendes neuronales Netzwerk, CNN). Diese werden hauptsächlich zur Bild- und Audioverarbeitung genutzt [67, 105]. Auch zur Klassifikation können CNN eingesetzt werden [77, 79, 83, 87]. Diese werden hier aber nicht betrachtet. Zusätzlich zu den künstlich neuronalen Netzwerken wird das Random Forest Verfahren zur Klassifikation von Molekülen in Kapitel 9 eingesetzt. Das Verfahren wird daraufhin getestet, ob es auch bei kleinen Eingaben der Lernmenge gute Vorhersagen bei der Klassifikation von Molekülen treffen kann. Das Random Forest Verfahren ist ein statistisches Verfahren,

genauer eine Ensemblemethode, die eine endliche Menge an unterschiedlichen Klassifikationsalgorithmen nutzt, um bessere Vorhersagen bei der Klassifikation oder Regression zu generieren [15].

2.5.1 *Künstlich neuronale Netzwerke*

Jedes Neuron im Netzwerkmodell von McCulloch und Pitts kann wie in Abbildung 16 implementiert werden. Künstliche Neuronen liefern in der Regel kontinuierliche Outputs und führen einfache Funktionen, wie z. B. das Sammeln von Informationen, aus. Unter Berücksichtigung ihrer Aktivierungsfunktionen (siehe Abschnitt 2.5.2.2) erzeugen sie einen Output. Das Netzwerk kann aus einem oder mehreren Inputknoten x_i bestehen. Inputknoten sind die Signale oder Muster, die aus einer externen Umgebung stammen und die Werte darstellen, die von den Variablen einer bestimmten Anwendung angenommen werden können. Die Eingangssignale werden normalisiert, um die Recheneffizienz von Lernalgorithmen deutlich zu verbessern. Die Relevanz eines jeden Inputknotens wird mit dem dazugehörigen Gewicht w_i multipliziert, somit werden alle externen Informationen gewichtet. Die synaptischen Gewichte $w_1, w_2, \ldots, w_n$ sind die Werte, mit denen jede der Eingangsvariablen gewichtet werden, wodurch ihre Relevanz bezüglich der Funktionalität des Neurons quantifiziert werden kann.

Der lineare Aggregator Σ sammelt alle gewichteten Eingangssignale, um eine Aktivierungsspannung zu erzeugen. Der Bias Θ ist eine Variable, die dazu verwendet wird, um den richtigen Grenzwert anzugeben. Somit kann das vom linearen Aggregator erzeugte Ergebnis einen Prüfwert für die Neuronenausgabe erzeugen. Die Ausgabe u stellt die gewichtete Summe seiner Eingaben dar, d. h. das Aktivierungspotential u ist das Ergebnis, das sich aus der Differenz zwischen dem linearen Aggregator und dem Bias ergibt. Falls dieser Wert positiv ist, d. h. falls $u \geqslant \Theta$ ist, dann erzeugt das Neuron ein stimulierendes, andernfalls ein hemmendes Potential. Das Ziel der Aktivierungsfunktion g ist die Begrenzung der Neuronenausgabe innerhalb eines Wertebereichs. Außerdem wird mit ihr eine Nichtlinearität erzeugt, falls $g(\cdot)$ nichtlinear ist. Das Ausgangssignal y gibt den Endwert, der von dem Neuron bei einer bestimmten Menge an Eingangssignalen erzeugt wird, an [33] (siehe Abbildung 16).

Die zwei folgenden Ausdrücke stellen das Ergebnis des von McCulloch und Pitts vorgeschlagenen künstlichen Neuronen Modells dar:

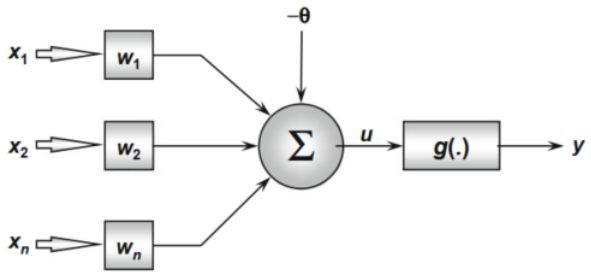


Abbildung 16: Neuronales Perzeptron von McCulloch und Pitts
Jedes Neuron im Netzwerkmodell von McCulloch und Pitts kann wie in der obigen Abbildung implementiert werden [33].

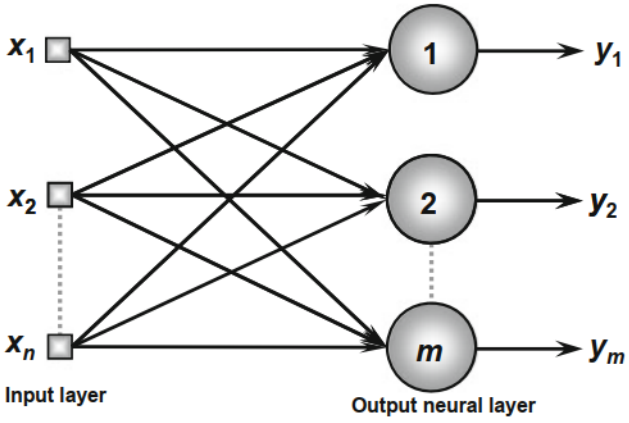


Abbildung 17: Einschichtiges feedforward-Netz
Ein einschichtiges feedforward-Netz, bestehend aus n Eingabe- und m Ausgabeknoten [33].

$$u = \sum_{i=1}^{n} w_i \cdot x_i - \Theta, \tag{2.5.1}$$

$$y = g(u). \tag{2.5.2}$$

Gleichung (2.5.1) summiert die gewichteten Eingangsinformation auf und zieht davon den Bias ab. Unter Verwendung einer geeigneten Aktivierungsfunktion wird die Neuronenausgabe limitiert. Die Ausgabe wird unter Verwendung der neuronalen Aktivierungsfunktion im Aktivierungspotential angewendet.

Ein neuronales Netz besteht in der Regel aus einer Eingabeschicht (Input-Layer), eventuell aus mehreren versteckten Schichten (Hidden-Layer) und einer Ausgabeschicht (Output-Layer). Die Anzahl der

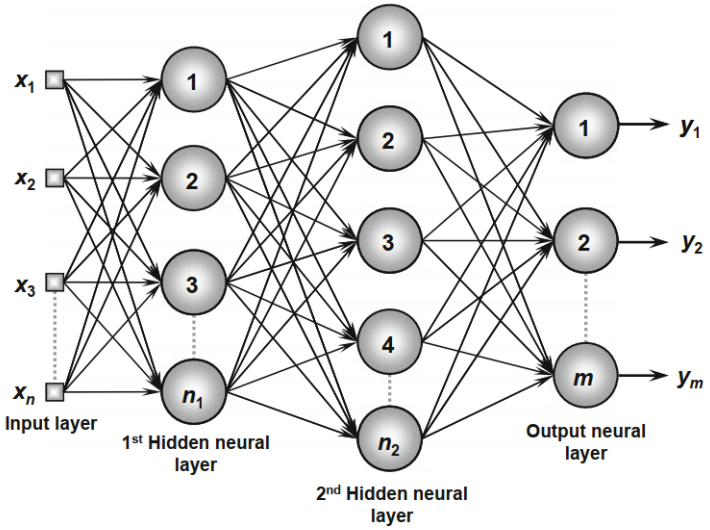


Abbildung 18: Mehrschichtiges feedforward-Netz
Die Abbildung stellt ein vollständig vernetztes MLP dar [33].

Neuronen in der Ausgabeschicht hängt von der jeweiligen Problemstellung ab. In der Regel existieren genau so viele Neuronen in der Ausgabeschicht, wie die Anzahl an Klassen zur Klassifizierung.

Obwohl die ersten Artikel zu künstlichen neuronalen Netzen (KNN) bereits früh veröffentlicht wurden, ist das Thema erst Anfang der 90er Jahre intensiv erforscht worden und weist aktuell noch ein enormes Forschungspotenzial auf [7, 14, 99, 104]. Die Anwendungen, bei denen KNNs eingesetzt werden, decken ein breites Spektrum ab. Darunter unter anderem die Analyse von Bildern [86], die Sprach- und Musterklassifizierung [86, 122] und unter anderem die Gesichtserkennung [8]. Basierend auf einem genetischen Profil einer bestimmten Person kann ein KNN z. B. auch zur Klassifizierung von Krebserkrankungen eingesetzt werden [31]. Im pharmazeutischen Bereich kann unter Zuhilfenahme von KNNs die Arzneimittelforschung unterstützt werden [21].

2.5.2 *Rekurrente neuronale Netzwerke*

Es existieren zahlreiche KNN-Modelle, einschichtige- und mehrschichtige feedforward-Netze und rekurrente-Netze (RNN) (siehe Abbildungen 17, 18 und 19). Feedforward-Netze leiten Informationen nur in eine Richtung und zwar vorwärts vom Eingabeknoten zu den Ausgabeknoten weiter. Einschichtige feedforward-Netze bestehen aus einer Ein- und Ausgabeschicht. Diese Netzwerke werden in der Regel zur Klassifizierung von Pattern (dt. Muster) und linearen Filterungsproblemen, wie der Signalverarbeitung, eingesetzt [25].

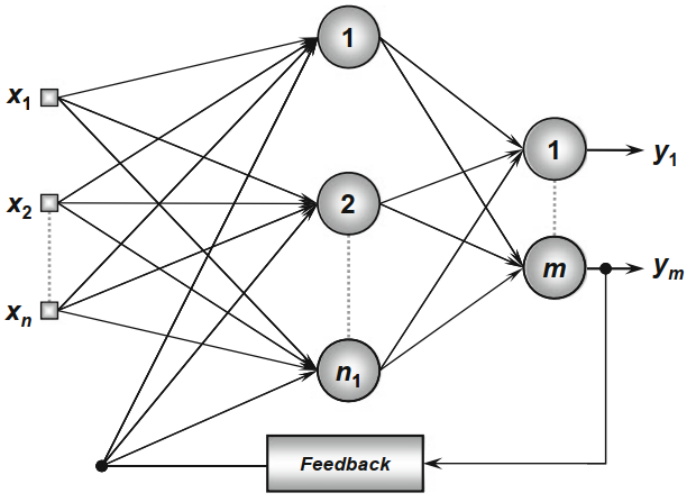


Abbildung 19: Rekurrentes Neuronales Netzwerk

Rekurrente Netze besitzen im Gegensatz zu den feedforward-gesteuerten Netzen auch rückgerichtete Kanten, um die Dynamik des Netzes zu erhöhen [33].

Mehrschichtige Netze hingegen, auch Multilayer Perceptrons (MLP) genannt, können neben einer Ein- und Ausgabeschicht aus weiteren Schichten, den sogennanten Hidden-Layers, bestehen. Sie werden z. B. bei der Funktionsapproximation, Patternklassifizierung, Prozesssteuerung und Optimierung eingesetzt [72]. MLP können sowohl feedforward-Netze als auch rekurrente Netze darstellen. RNN können im Gegensatz zu den feedforward-Netzen, zusätzlich zu den vorwärts auch rückwärts gerichtete Kanten aufweisen, sodass die Ausgaben eines Neurons als Eingaben für Neuronen in einer zuvor- oder der gleichliegenden Schicht dienen können. Dieses Vorgehen erhöht die Dynamik des zugrundeliegenden neuronalen Netzwerkes. Dadurch ist es möglich, Informationen über die vorherigen Eingaben zu speichern. Die in dieser Arbeit genutzten Netzwerke sind alle vollständig vernetzt (siehe Abbildung 18) [33].

RNNs gehören zur Familie der neuronalen Netzwerke zur Verarbeitung sequentieller Daten variabler Länge. In RNNs ist es möglich, Parameter über verschiedene Teile eines Modells hinweg zu nutzen. Die Parameterfreigabe zwischen unterschiedlichen Teilen des RNN ermöglicht es, die Modelle auf unterschiedliche Probleme anzupassen, sie zu erweitern und zu generalisieren. Falls man für jeden Wert eines Zeitindex einen eigenen Parameter hätte, könnte man Sequenzlängen, die während des Trainings beobachtet wurden, nicht generalisieren oder ihre statistische Stärke über die unterschiedlichen Sequenzlängen über diverse Positionen im Netzwerk geteilt werden. Ein solches Teilen ist besonders wichtig, falls eine bestimmte Infor-

mation an mehreren Stellen innerhalb einer Sequenz vorkommt. Angenommen, es wurde ein feedforward-Netzwerk trainiert, das Sätze (Wortfolgen) fester Länge verarbeiten kann, dann hätte ein herkömmliches, vollständig verbundenes feedforward-Netzwerk für jedes Eingabemerkmal separate Parameter. Daher müssten alle Regeln der Sprache für diesen Satz an jeder Position die Parameter separat erlernt werden. Im Vergleich dazu teilt ein RNN die gleichgewichteten Parameter über mehrere Zeitschritte hinweg mit dem Netzwerk [59].

2.5.2.1 *Long Short-Term Memory*

Zum gegenwärtigen Zeitpunkt werden die effektivsten Sequenzmodelle, die in praktischen Anwendungen Verwendung finden, als gated RNNs bezeichnet. Dazu gehören die LSTM-Netzwerke [59]. Long short-term memory (dt. langes Kurzzeitgedächtnis, LSTM) ist ein Verfahren zur Verbesserung der RNN-Modelle. Das RNN erhält durch das LSTM ein „langes Kurzzeitgedächtnis“ [68]. Das Problem, das bei RNNs ohne LSTM auftreten kann, ist folgendes: Je mehr Schichten ein RNN aufweist und je länger es trainiert wurde, desto komplexer kann das Netzwerk werden, infolge dessen gehen bestimmte Informationen und Erfahrungen verloren. Die LSTM-Technik umgeht dieses Problem, indem sie eine Art langes Kurzzeitgedächtnis schafft. Das Verfahren wurde 1997 von Hochreiter et al. [68] entwickelt, wird fortlaufend erweitert und ist auf den unterschiedlichsten Problemstellungen anwendbar [59, 102].

Eine LSTM-Zelle besteht aus vier grundlegenden Bausteinen, dem Datenspeicher, der Eingabe-, der Vergessen- und der Ausgabeschicht. Im oberen Abschnitt der Abbildung 21 ist der Datenspeicher einer LSTM dargestellt. Dieser besteht aus einem Speicherblock C_{t-1} einer vorherigen LSTM und einem Speicherblock der aktuellen Zeitperiode C_t. Es existieren drei unterschiedliche Schichten in einer LSTM-Zelle, um den Datenspeicher zu steuern. Jede dieser Schichten gibt an, in welchem Umfang Informationen weitergeleitet werden. Sie bestehen jeweils aus einer Sigmoid-Schicht gefolgt von einer punktweisen Multiplikation (siehe gelbe Rechtecke und rosa Kreise in Abbildung 20). Eine Sigmoid-Aktivierungsfunktion (kurz Sigmoid) gibt Zahlenwerte zwischen $[0, 1]$ aus, d. h. bei einem Wert von 0 lässt sie keine Informationen, bei einem Wert von 1 alle durch.

Die Sigmoid der Vergessensschicht entscheidet am Anfang unter Berücksichtigung des Outputs der vorherigen LSTM h_{t-1} und einem Input-Vektor x_t, welche Daten in der aktuellen LSTM erhalten bleiben und welche verworfen werden.

Im nächsten Schritt entscheidet die Sigmoid der Eingabeschicht, welche Werte und in welchem Umfang die Daten aktualisiert wer-

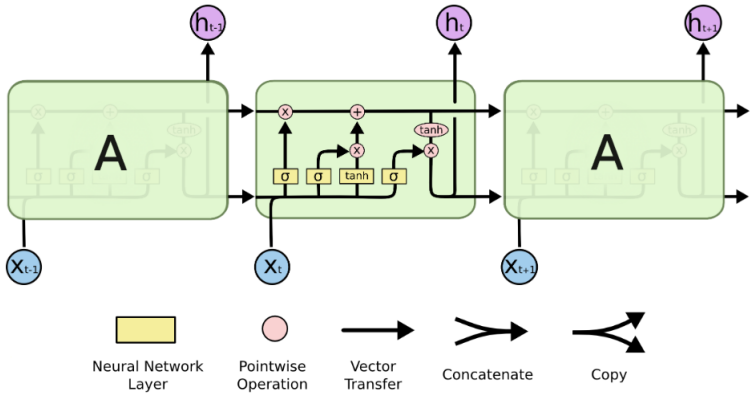


Abbildung 20: Aufbau einer LSTM-Zelle
Übersicht der LSTM-Technik zur Verbesserung eines RNN-Modells. x_t stellt den Eingabe- und h_t den Ausgabeknoten im Zeitpunkt t dar. In der obigen Abbildung überträgt jeder Pfeil einen kompletten Vektor an Daten. Die rosa Kreise repräsentieren punktweise Operationen, wie z. B. eine Vektoraddition, während die gelben Kästchen angelernte neuronale Netzwerkschichten sind. Zusammenführende Pfeile bezeichnen die Verkettung von Daten, während die gegabelten Pfeile Daten kopieren und an unterschiedliche Speicherorte weiterleiten [102].

den. Die Hyperbeltangens-Aktivierungsfunktion (kurz $\tanh$) erzeugt einen Vektor an neuen Wertekandidaten, die zum Speicher hinzugefügt werden könnten. Darauf folgend werden die Informationen aus der Sigmoid, der Eingabeschicht und den Wertekandidaten aus der $\tanh$-Aktivierungsfunktion kombiniert, um den aktuellen Speicherzustand C_t zu aktualisieren.

Die Sigmoid der Ausgabeschicht legt im letzten Schritt fest, welche Informationen des Speichers ausgegeben werden. Diese Werte werden mit der $\tanh$-Aktivierungsfunktion multipliziert, um nur die gewünschten Informationen weiterzugeben [102].

2.5.2.2 *Aktivierungsfunktionen*

Aktivierungsfunktionen können in zwei Gruppen unterteilt werden, in partiell und total differenzierbare Funktionen. Partielle Aktivierungsfunktionen weisen Punkte auf, die keine erste Ableitung besitzen. Total differenzierbare Aktivierungsfunktionen hingegen weisen für alle Punkte ihres Definitionsbereichs Ableitungen erster Ordnung auf. Alle in KNNs eingesetzten Funktionen sind im gesamten Definitionsbereich total differenzierbar. Die Wahl der Aktivierungsfunktionen gibt den späteren erzeugten Ausgabewert an. Es existieren hauptsächlich vier unterschiedliche Funktionsarten, die in KNNs ein-

gesetzt werden. Diese sind die logistische-, Hyperbeltangens-, Gauß- und Linearfunktion. Eine logistische Funktion erzeugt immer reelle Werte zwischen 0 und 1, während eine Hyperbeltangensfunktion hingegen immer reele Werte zwischen -1 und 1 annimmt. Sowohl die logistischen- als auch die Hyperbeltangensfunktionen zählen zur Klasse der Sigmoid-Funktionen [33].

Es existieren unterschiedliche KNN-Aktivierungsfunktionen. Die am häufigsten genutzten werden im Folgenden kurz vorgestellt [34]. Die linearen Aktivierungsfunktionen erzeugen Outputs, die dem Aktivierungspotential entsprechen. Die Hyperbeltangensaktivierungsfunktion berechnet elementweise den $\tanh(x)$ und die Sigmoid-Aktivierungsfunktion den Sigmoid von x. Die Rectified Linear Unit (ReLU) hat sich bei vielen Problemstellungen als vorteilhaft erwiesen [11]. Sie gibt nur Werte aus, die größer gleich 0 sind. Sie ist wie folgt definiert:

$$g(x) = \max(0, x).$$

Zur Klassifizierung von Daten sind nicht alle genannten Funktionen nützlich [126]. Die Softmax-Funktion z. B. kann nur mit zwei Klassen umgehen. Der Output eines jeden Wertes liegt in der Softmaxfunktion zwischen 0 und 1, ähnlich der Sigmoidfunktion. Sie teilt aber jeden Output so auf, dass der Gesamtoutput gleich 1 ist. Der Output der Softmax-Funktion $\sigma(z)$ entspricht einer kategorialen Wahrscheinlichkeitsverteilung. Sie ist nachfolgend dargestellt, wobei z ein Inputvektor zur Ausgabeschicht darstellt. Die Ausgabeeinheiten werden mit j bezeichnet [126].

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}.$$

2.5.2.3 *Trainings- und Lernprozess*

Eine der wichtigsten Aufgaben von KNNs ist, aus einer Eingabe von Beispielen/Pattern, die das Systemverhalten wiedergeben, zu lernen. Nachdem das Netzwerk die Beziehung zwischen den Ein- und Ausgabedaten erlernt hat, sollte es Lösungen verallgemeinern können, die nahe an der erwarteten oder gewünschten Ausgabe bei beliebigen Inputwerten liegen. Daher besteht der Trainingsprozess eines KNNs darin, die Abstimmung der synaptischen Gewichte und dessen Grenzwerte der Neuronen anzupassen. Die Kostenfunktion im KNN gibt die Abweichungen zwischen den Ein- und Ausgabedaten

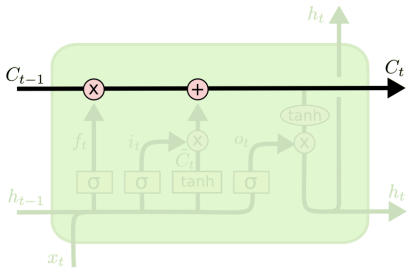


Abbildung 21: Der Datenspeicher einer LSTM-Zelle
C_{t-1} stellt einen Speicherblock aus einer vorherigen LSTM-Zelle und C_t den der aktuellen Zeitperiode dar. Die rosa Kreise repräsentieren punktweise Operationen, wie z. B. Vektoradditionen [102].

an. Den Wert der *Kostenfunktion* gilt es zu minimieren. Zur Klassifikation von Daten werden in der Regel crossentropy (dt. Kreuzentropie), für Regressionen der mean squared error (dt. mittlere quadratische Abweichung, MSE) eingesetzt (siehe [59]). Die Anzahl an Schritten zum Trainieren eines KNN wird als Lernalgorithmus bezeichnet. Der Lernalgorithmus in einem MLP nennt sich Backpropagation (siehe Abschnitt 2.5.2.7)

Der komplette Datensatz der zur Verfügung stehenden Beispiele/Pattern, die das Systemverhalten enthalten, werden in der Regel in zwei Mengen aufgeteilt, in Trainings- und Testmenge. Zum Trainieren des KNNs werden in der Regel circa 60–90% der Stichproben der gesamten Datenmenge genutzt, den Rest nutzt man als Testmenge. Hierbei wird validiert, ob die Netzwerkfähigkeiten von Generalisierungslösungen im akzeptablen Bereich liegen, sodass eine gegebene Topologie verifiziert werden kann. Bei der Dimensionierung dieser Teilmengen müssen jedoch auch statistische Merkmale der Daten berücksichtigt werden. Während des Trainingsprozesses von KNNs wird jede vollständige Präsentation aller zum Trainingssatz gehörenden Beispiele zur Anpassung der synaptischen Gewichte und Grenzwerte als Trainingsepoche (kurz Epoche) bezeichnet.

2.5.2.4 *Überwachtes Lernen*

Das überwachte Lernen (engl. supervised learning) ist das am häufigsten verwendete und erfolgreichste Modell des maschinellen Lernens [100]. Es wird immer dann eingesetzt, wenn man aus einer gegebenen Eingabe ein bestimmtes Ergebnis vorhersagen möchte, das bereits a priori bekannt ist. Das Ziel ist es, eine möglichst genaue Vorhersage auf Daten zu treffen, die das KNN noch nicht gesehen hat. Das KNN vergleicht fortwährend die Ein- und Ausgabedaten und passt die synaptischen Gewichte und den Bias so an, dass für jede Eingabe möglichst die gewünschte Ausgabe erfolgt [33].

2.5.2.5 *Unbeaufsichtigtes Lernen*

Zusätzlich zum überwachten Lernen existiert noch das unbeaufsichtigte Lernen (engl. unsupervised learning). Im Gegensatz zum überwachten Lernen muss das KNN hier keine Kenntnisse bezüglich der Ausgabewerte besitzen. Das KNN muss aus den Elementen der Eingabedaten selbständig Gemeinsamkeiten identifizieren. Das KNN passt die synaptischen Gewichte und den Bias des Netzwerks stetig so an, dass ein Clustering innerhalb des Netzwerks ermöglicht wird. Alternativ kann die maximale Anzahl an möglichen Clustern bereits vorab im KNN angeben werden [33].

2.5.2.6 *Lernrate*

Um die synaptischen Gewichte in einem KNN während der Trainingsphase anzupassen, werden Lernregeln benötigt, die angeben, wie die Gewichte an die vorhandenen Daten angepasst werden sollen, um den gewünschten Output zu erzeugen [107]. Die Lernrate η bestimmt, wie schnell der Trainingsprozess zur Konvergenz führt. Die Wahl der Lernrate sollte sorgfältig getroffen werden, um Unausgewogenheiten im Trainingsprozess zu vermeiden. Der Wertebereich der Lernrate liegt in $[0, 1]$ [33].

Es existiert leider keine optimale Lernrate für alle Arten an KNNs. Für jedes KNN muss die Lernrate angepasst werden. Eine höhere Lernrate bewirkt, dass die Sprünge in der Hyperebene größer werden. Vom Startpunkt werden weit entfernte Minima schneller erreicht, globale Minima aber häufiger übersprungen. Eine niedrigere Lernrate führt dazu, dass kleinere Schritte beim gradient descent (dt. Gradientenabstiegsverfahren, GD, siehe Abschnitt 2.5.2.7) berechnet werden. Globale Minima werden aber nicht mehr so leicht übersprungen, die Trainingszeit kann bis zum Erreichen eines Minimums sehr teuer werden [107].

Die Wahl der richtigen Lernrate kann sich als äußerst schwierig erweisen. Eine zu geringe Lernrate führt zu einer sehr langsamen Konvergenz, eine zu hohe kann die Konvergenz behindern und führt dazu, dass die Verlustfunktion um das Minimum schwankt oder sogar divergiert. Es kann von Vorteil sein, die Lernrate während des Trainings anzupassen [111].

2.5.2.7 *Backpropagation und Gradientenabstiegsverfahren*

Ein feedforward-Netzwerk erzeugt aus einer Eingabe x die Ausgabe y, dabei fließen die Informationen vorwärtsgerichtet durch das Netzwerk. Hierbei enthält x die initialen Informationen, die durch alle Ebenen des Netzwerks propagieren und letztendlich die Ausgabe y erzeugen. Dieses Verfahren wird forward-propagation (Vorwärtsausbreitung) genannt. Unter Zuhilfenahme der Kostenfunktion

wird die Abweichung zwischen den Ausgabe- und Eingabewerten berechnet. Die Abweichung wird im Netzwerk als Fehler interpretiert. Das Backpropagation-Verfahren (dt. Fehlerrückführung) propagiert diesen Fehler von der Ausgabe- zurück zur Eingabeschicht, um die Gewichte und den Bias anzupassen [59].

Das Backpropagation-Verfahren ist ein Gradientenabstiegsverfahren (GD), das versucht, die Abweichung als Ergebnis der Kostenfunktion zu minimieren. GDs werden generell dazu verwendet, um Maxima oder Minima von Funktionen zu berechnen und sind iterative Verfahren zur Optimierung einer differenzierbaren Funktion, die als Näherungswert für den Gradienten bestimmt wird. Hierbei ist der Gradient ein Vektor, der für jeden differenzierbaren Punkt einer Funktion definiert ist. Der Gradient gibt hierbei die Richtung der größten Steigung an. In die negative Richtung werden die Gewichte und der Bias im Netzwerk in jedem Durchlauf angepasst, um die Kostenfunktion des Modells zu minimieren. Das Ziel der Optimierung ist es, statt eines lokalen das globale Minimum zu erreichen.

GDs sind Optimierungsverfahren, die auch Fehler aufweisen können, da man nicht immer anhand des Ergebnisses ersehen kann, ob ein Fehler berechnet wurde. Ein GD kann zum Beispiel in einem lokalen Minimum hängen bleiben [82].

Mit dem GD ist es möglich, die Zielfunktion zu minimieren. Dabei bestimmt die Lernrate die Anzahl an Schritten zu einem (lokalen) Minimum.

Es existieren derzeit drei Varianten an GDs, um die Gewichte und den Bias zu aktualisieren, das batch gradient decent (dt. Stapel-Gradientenabstiegsverfahren, BGD), das stochastic gradient descent (dt. Stochastische-Gradientenabstiegsverfahren, SGD) und mini batch gradient descent (dt. Ministapel-Gradientenabstiegsverfahren, MBGD).

Das BGD berechnet den Gradienten der Kostenfunktion für das komplette Trainingsset. Da alle Gradienten gleichzeitig für die komplette Datenmenge berechnet werden, um ein Update zu erzeugen, ist das Verfahren bei einer großen Datenmenge sehr träge und unlösbar, falls die Datenmenge nicht in den Speicher passt. Doch das BGD-Verfahren konvergiert für konvexe Gebiete in das globale und für nicht-konvexe in einem lokalen Minimum.

Das SGD-Verfahren erzeugt für jedes Element in der Trainingsmenge ein Parameterupdate. Das BGD-Verfahren hingegen führt redundante Berechnungen für große Datensätze durch, da es Gradienten vor jeder Parameteraktualisierung für ähnliche Beispiele neu berechnet. Das SGD-Verfahren verzichtet auf diese redundante Berechnungen, indem es jeweils ein Update ausführt. Aus diesem Grund ist

das SGD-Verfahren in der Regel bedeutend schneller und kann auch zum Online-Lernen verwendet werden. SGDs führen häufige Aktualisierungen mit einer hohen Varianz durch, die dazu führen, dass die Zielfunktion stark schwankt.

Während das BDG-Verfahren zum Minimum konvergiert, ermöglicht das Fluktuieren des SGD-Verfahrens in der Regel bessere lokale Minimums aufzufinden. Aus diesem Grund kann es sein, dass das SDG über das exakte Minimum hinausschießt. Doch es hat sich herausgestellt, dass das SDG-Verfahren ein ähnliches Konvergenzverhalten aufweist wie das BDG-Verfahren, falls die Lernrate langsam im SGD-Verfahren reduziert wird,

Das MBDG-Verfahren verbindet die BDG- und SDG-Verfahren, indem es ein Update für jedes mini-batch aus n Elementen der Trainingsmenge berechnet. Dadurch wird die Varianz der Parameteraktualisierungen reduziert und kann zu einer stabileren Konvergenz führen. Generell liegt die mini-batch Größe zwischen 50 und 256 Elementen, sollte aber auf die jeweilige Problemstellung angepasst werden. Das MBGD garantiert in der Regel keine gute Konvergenz, bietet aber einige Vorteile und Herausforderungen.

2.5.2.8 *Optimierer für das Gradientenabstiegsverfahren*

Es existieren zurzeit zahlreiche Optimierer für das Gradientenabstiegsverfahren. Im folgenden Abschnitt werden die aktuell am häufigsten eingesetzten vorgestellt [111, 113].

Adagrad ist ein Optimierer, der für jeden Parameter eine andere Lernrate nutzt. Die Adagrad-Aktualisierungsregel modifiziert die Lernrate in jedem Zeitschritt und für jeden Parameter, basierend auf den bereits berechneten Gradienten. Adagrad führt größere Aktualisierungen für selten und kleinere Aktualisierungen für häufiger genutzte Parameter durch. Daher arbeitet der Optimierer auch gut auf verstreuten Daten. Adagrad hat die Robustheit von SGDs erheblich verbessert [111]. Darüber hinaus kann das Verfahren für das Trainieren großer KNNs genutzt werden. Adagrad besitzt vor allem den Vorteil, dass die Lernrate nicht manuell angepasst werden muss. Der Nachteil des Optimierers ist, dass die Lernrate während des Trainings abnimmt und beliebig klein werden kann. Infolge dessen erhält das Verfahren keine zusätzlichen Informationen. Die folgenden Algorithmen versuchen diesen Fehler zu beheben.

Adadelta ist eine Adagrad Erweiterung, die die Nachteile von Adagrad minimieren möchte. Anstatt alle vorherigen quadrierten Gradienten anzusammeln, beschränkt Adadelta die zu akkumulierenden vorherigen Gradienten auf eine bestimmte Größe. Anstatt die vorherigen Gradienten ineffizient zu speichern, wird die Summe der Gra-

dienten rekursiv als abnehmender Durchschnitt aller vorherigen Gradienten definiert. Der laufende Durchschnitt im Zeitschritt t hängt in diesem Fall nur vom vorherigen Durchschnitt und dem aktuellen Gradienten ab.

RMSprop wurde unabhängig davon, aber im gleichen Zeitraum entwickelt wie das Adadelta-Verfahren, um ebenfalls die Schwächen des Adagrad-Optimierers zu minimieren. Der erste Adadelta Aktualisierungsvektor ist identisch mit dem des RMSprop-Verfahrens. RMSprop dividiert die Lernrate durch einen exponentiell abnehmenden Durchschnitt der quadrierten Gradienten.

Adam ist ein Optimierer, der eine adaptive Lernrate für jeden Parameter berechnet. Neben der Speicherung eines exponentiell geglätteten Mittelwerts aus den vorherigen quadratischen Gradienten speichert das Verfahren auch den exponentiell geglätteten Mittelwert der vorangegangen Gradienten, ähnlich eines Momentums. Das Momentum trägt dazu bei, dass SGD-Verfahren schneller in die richtige Richtung gelenkt werden. Hierzu wird ein Bruchteil des letzten Aktualisierungsvektors zum aktuellen hinzugefügt. Der exponentiell geglättete Mittelwert teilt den Daten einer Zeitreihe exponentiell abnehmende Gewichte zu. Somit werden aktuellere Daten stärker gewichtet als ältere [113].

Adagrad, Adadelta und RMSprop-Optimierer konvergieren ähnlich schnell. Sie und der Adam-Optimierer sind in der Regel sehr gut auf unterschiedliche Problemstellungen anwendbar und bieten generell eine gute Konvergenz [111].

2.5.2.9 *Over-/Underfitting*

Die zentrale Herausforderung beim maschinellen Lernen besteht darin, dass der Algorithmus auch auf unbekannten Eingaben eine gute Vorhersage bietet, wie z. B. mit hoher Wahrscheinlichkeit ein bestimmtes Element zu seiner Klasse zuzuordnen, ohne es vorher gesehen zu haben [59].

Beim Klassifizieren von Problemen kann es vorkommen, dass die Validierung der Testdaten während des Trainings eine Zeit lang steigt (Anzahl Epochen oder Durchläufe), den Höchstwert erreicht und dann plötzlich abfällt. Tritt dieser Fall ein, wird von Overfitting (dt. Überanpassung des Modells auf die vorhandenen Daten) gesprochen. Das KNN hat sozusagen die Daten auswendig gelernt. In diesem Fall ist es nur in einem gewissen Maße fähig mit Daten umzugehen, die es noch nicht gesehen hat, d. h. es kann ungesehene Daten nicht richtig klassifizieren. Es ist wichtig, dass sich das trainierte Wissen auf unbekannte Daten generalisieren lässt.

Wird das Modell jedoch zu lange trainiert, overfittet es und erkennt in der Regel Pattern, die nicht auf die Testdaten generalisierbar sind (siehe Abbildung 22). Das Overfitting kann vermieden werden, indem zum Trainieren Daten ausgewählt werden, von denen man z. B. a priori weiß, zu welcher Klasse sie gehören. Eine weitere Möglichkeit kann durch die Regularisierung verhindert werden. Dadurch werden Einschränkungen hinsichtlich der Art und Menge der Daten festgelegt, die das Modell speichern kann. Dies wird erreicht, indem die Gewichte im Netzwerk dazu gezwungen werden, kleine Werte anzunehmen. So erreicht man eine gleichmäßigere Gewichtsverteilung im Modell. Falls das Netzwerk in der Lage ist, nur wenige Daten zu speichern, wird es durch den Optimierungsprozess gezwungen, sich auf die wichtigsten Merkmale zu beschränken. Somit sollte es dem KNN möglich sein, die Trainingsdaten besser zu verallgemeinern. Dieses Verfahren wird auch als Gewichtungsregulierung bezeichnet. Hierbei wird der Kostenfunktion ein Regularisierungsterm (Strafterm) hinzugefügt, d. h. je höher die Gewichte im Netzwerk werden, desto höher sind die Strafkosten. Es existieren derzeit unterschiedliche Regularisierungen. Die L1-Regularisierung bestraft die Gewichte proportional zur Summe der absoluten Werte. Hierbei werden Gewichtungen irrelevanter oder kaum relevanter Merkmale auf 0 gesetzt und somit aus dem Modell entfernt. Bei der L2-Regularisierung werden die Gewichte proportional zur Summe der quadrierten Gewichte bestraft. Die L2-Regularisierung hilft, Ausreißer nahe 0 zu bringen. Die L2-Regularisierung verbessert in linearen Modellen generell die Generalisierung [5].

Eine weitere Möglichkeit, Overfitting zu vermeiden, ist die Anzahl der zu lernenden Parameter im KNN zu reduzieren, so kann es Daten nicht nur auswendig lernen. Wird das Modell zu klein gewählt, wird es schwierig sein, dass das Modell aus den Trainingsdaten die wichtigen Merkmale herausfiltern kann. Aus diesem Grund sollte ein gutes Gleichgewicht zwischen „zu viel Kapazität“ und „nicht genügend Kapazität“ im KNN-Modell existieren.

Dropout ist eine der effektivsten und am häufigsten verwendeten Regularisierungstechniken in KNNs. Das Dropout-Verfahren deaktiviert eine gewisse Anzahl an Neuronen in einem Layer während des Trainings. Die Dropoutrate bestimmt den Bruchteil der Features, die auf Null gesetzt werden und hat in der Regel einen Wert zwischen $0,2$ und $0,5$ [5].

Underfitting tritt ein (dt. Unteranpassung), falls die Auswahl der Testdaten noch verbessert werden kann. Dies könnte aber auch andere Gründe haben, wie z. B. dass das Modell nicht leistungsfähig genug, übermäßig reguliert ist oder einfach nicht lange genug trainiert wurde. Dies bedeutet, dass das Netzwerk die relevanten Pattern in den Trainingsdaten nicht gelernt / erkannt hat.

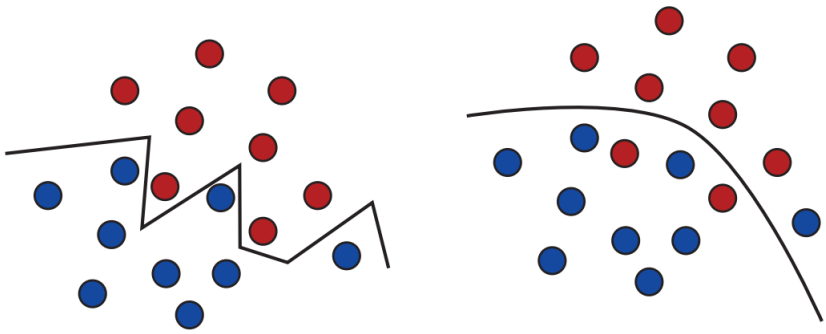


Abbildung 22: Overfitting
In der linken Darstellung ist zu erkennen, dass alle Samplepunkte (dt. Beispielpunkte) im Trainingsset getrennt wurden. Die Trennline, d. h. das zugrunde liegende Modell wurde zu stark an das Trainingsset angepasst. Die Wahrscheinlichkeit, dass das Modell auf eine andere Datenmenge verallgemeinert werden kann, ist nicht sehr hoch. Im Gegensatz dazu ist in der rechten Abbildung zu sehen, dass das Modell trotz falscher Klassifizierung einiger Punkte der Trainingsstichprobe auch auf anderen Eingaben verallgemeinert werden kann, um Daten richtig voneinander zu trennen. Es würde auf ungesehenen Daten besser generalisieren [99].

2.5.3 *Random Forest*

Die Beschreibung des Random Forest Ansatzes wurde, soweit nicht anders angegeben, aus dem Buch von Friedman et al. [54] entnommen.

Das zugrundeliegende Verfahren beim Random Forest Ansatz sind Entscheidungsbäume (siehe Abschnitt 2.5.3.1). Bei der Klassifikation werden B Entscheidungsbäume miteinander kombiniert. Jeder Baum in diesem Wald darf hierbei eine Entscheidung bei der Klassifikation eines vorliegenden Eingabeelementes treffen, d. h. jeder Baum im Wald sagt für ein bestimmtes Element genau eine Klasse voraus. Das Random Forest Verfahren prognostiziert nun anhand der am häufigsten getroffenen Entscheidungen, zu welcher Menge ein bestimmtes Element gehört und gibt diese aus, d. h. wurden z. B. 1.000 Bäume zur Klassifizierung genutzt und 501 davon sagen für ein Element die Klasse c voraus, wird dieses Element dieser zugeordnet. Das Random Forest Verfahren besteht aus unkorrelierten Entscheidungsbäumen, die mittels Bagging oder Boosting (siehe Abschnitt 2.5.3.4) randomisiert erzeugt werden. Wie bei den RNN-Modellen ist die Wahl der Hyperparameter entscheidend, wie z. B. die Anzahl an Bäumen, die Baumtiefe und die Art der genutzten Bäume. Hier können Klassifizierungs-

(Abschnitt 2.5.3.3) und Regressionsbäume (Abschnitt 2.5.3.2) eingesetzt werden.

Um die einzelnen Komponenten des Random Forest Verfahrens genauer beschreiben zu können, werden als Erstes Entscheidungsbäume, anschließend Klassifikations- und Regressionsbäume (CART, engl. classification and regression trees) und später die Verfahren Bagging und Boosting eingeführt und erläutert.

Das Random Forest Verfahren wurde von Breiman 2001 in [16] eingeführt. Es fügt dem Bagging Verfahren eine zusätzliche Zufälligkeitskomponente hinzu. Im Random Forest Verfahren wird bei der Konstruktion eines jeden Baums eine andere Bootstrap-Stichprobe der vorhandenen Daten genutzt. Das Bootstrap-Verfahren ist in der Statistik eine Methode des Resampling. Dabei werden Statistiken mehrmals auf der Grundlage einer einzelnen Stichprobe berechnet [46]. In Standardbäumen wird jeder Knoten dort aufgeteilt, wo es die beste Aufteilung unter allen Variablen gibt. Im Random Forest Verfahren wird jeder Knoten unter Verwendung der besten Teilmenge von Prädiktoren aufgeteilt, die zufällig an diesem Knoten ausgewählt wurde. Diese Strategie erweist sich im Vergleich zu vielen anderen Klassifikatoren, darunter unter anderem die neuronalen Netzwerke, als sehr gute Strategie, die gleichzeitig robust gegenüber Overfitting ist [16].

Aufbau und Beschreibung des Random-Forest-Algorithmus zur Klassifikation und Regression [89]:

1. Ziehe B-Bootstrap-Samples aus der Ursprungsdatenmenge
2. Erzeuge für jedes Bootstrap-Sample einen unbeschnittenen Klassifikations- oder Regressionsbaum mit den folgenden Modifikationen: Anstatt an jedem Knoten die beste Aufteilung unter allen Prediktoren auszuwählen, werden zufällig m_{try} Prediktoren aus der Probe entnommen und die beste Aufteilung unter diesen Variablen wird ausgewählt (das Bagging kann als Spezialfall des Random Forest Verfahrens angesehen werden, falls $m_{try} = p$ die Anzahl an Prädiktoren darstellt).
3. Vorhersage neuer Daten durch Aggregation der Vorhersagen der B-Bäume (d. h. Mehrheitsentscheid bei der Klassifizierung, Berechnung des Durchschnitts im Falle der Regression).

2.5.3.1 *Entscheidungsbäume*

Entscheidungsbäume sind im Prinzip nichts anderes als If-Else Abfragen, die dazu verwendet werden können, um Elemente einer Menge zu klassifizieren. Nachfolgend wird anhand eines Beispiels ein Entscheidungsbaum vorgestellt, der entscheidet, ob ein Passagier auf der Titanic überlebt hat (Hinweis: Das Beispiel dient nur zur Darstellung

und weist nicht die korrekten Daten der Überlebenden beim Untergang der Titanic auf). Der Entscheidungsbaum unterteilt die Eingabemenge in zwei Klassen („hat überlebt“ und „hat nicht überlebt“) anhand von Grenzwerten mehrerer Attribute auf (Geschlecht, Alter, Anzahl an Geschwistern). Dabei stellt jeder Grenzwert eine bestimmte Beschränkung für eine Menge an Elementen dar.

Die aus [96] entnommene Abbildung 23 zeigt am Beispiel der Passagiere der Titanic, ob jemand überlebt hat (grüner Punkt Ü definiert, ob eine Person überlebt hat; roter Punkt V die Klasse an Personen, die verstorben ist). Alle Elemente, die klassifiziert werden, befinden sich am Anfang in der Menge des Wurzelknotens. Nun werden die einzelnen Passagiere anhand des Merkmals Geschlecht in zwei Klassen unterteilt (weiblich, männlich). Alle Frauen scheinen nach diesem Entscheidungsbaum überlebt zu haben, hingegen nicht alle Männer. Nun wird für die Menge aller Männer, am Merkmal Alter, verzweigt. Alle, die älter als $9,5$ Jahre alt waren, sind ebenfalls verstorben. Die Männer, die jünger waren, werden nun an der Variable Anzahl an Geschwistern verzweigt. Alle männlichen Passagiere, die jünger als $9,5$ Jahre alt waren und mehr als $2,5$ Geschwister aufwiesen, starben ebenfalls, hingegen nicht die, die weniger als $2,5$ Geschwister aufwiesen, sie überlebten.

Algorithmen des maschinellen Lernens können, wie erläutert, in zwei Klassen unterteilt werden, dem überwachten und dem unbeaufsichtigtem Lernen. Ein Entscheidungsbaum ist ein überwachter Lernalgorithmus. Er hat eine baumartige Struktur mit einem Wurzelknoten. Der Algorithmus identifiziert die wahrscheinlichste Klasse für ein gegebenes Element anhand seiner Attribute/Variablen. Ein zu klassifizierendes Element wird von der Wurzel zu einem Blattknoten/Terminalknoten anhand seiner Attribute verzweigt und eingeordnet. Ein Blattknoten ist ein Knoten, der keine Nachfolgeknoten im Graphen aufweist [96].

2.5.3.2 *Regressionsbäume*

Die Klassifikations- und Regressionsbäume wurden 1984 von Breiman, Friedman, Olshen und Stone in [18] vorgestellt. Sie sind konzeptionell einfach und dennoch mächtig. Man betrachte ein Regressionsmodell mit einer Zielgröße Y und den Inputs X_1 und X_2, die jeweils Werte im Einheitsintervall $[0, 1]$ annehmen können. Der linke obere Teilausschnitt in Abbildung 24 zeigt eine Aufteilung des Merkmalsraums, die durch parallele Linien zu den jeweiligen Koordinatenachsen erzeugt werden. Ein Merkmalsraum ist ein mathematischer Raum, der ein Objekt durch seinen Messwert in Bezug auf dessen besondere Merkmale bestimmt [9]. In jedem Element der Partition kann Y mit einer anderen Konstanten modelliert werden. Obwohl jede Partitionierung durch eine einfache Beschreibung wie $X_1 = c$ definiert

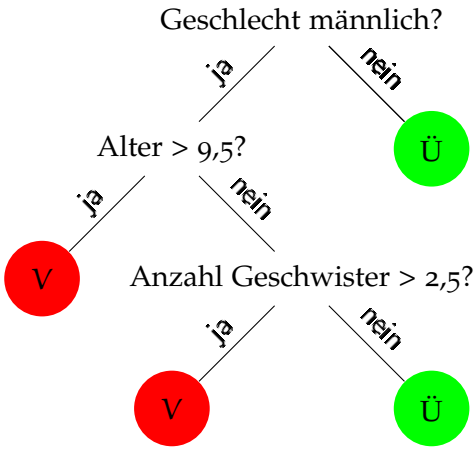


Abbildung 23: Entscheidungsbaum
Entscheidungsbaum am Beispiel der Titanic-Passagiere [96]

werden kann, sind einige der resultierenden Bereiche komplizierter zu bestimmen. Um die Beschreibung zu vereinfachen, wird das Beispiel auf die binäre Raumpartitionierung, wie im rechten oberen Teilausschnitt der Abbildung 24 dargestellt, beschränkt. Als Erstes wird der Raum in zwei Regionen aufgeteilt und modelliert dann den Rückgabewert Y aus dem Mittelwert einer jeden Region. Man wählt nun die Variable und den Aufteilungspunkt, um die beste Anpassung zu erreichen. Werden eine oder auch beide Regionen in zwei weitere Regionen aufgeteilt, wird dieser Prozess solange fortgesetzt, bis ein Abbruchkriterium angewendet werden kann. Im rechten oberen Teilausschnitt der Abbildung 24 wird der Raum zunächst bei $X_1 = t_1$ aufgeteilt. Dann wird die Region $X_1 \leqslant t_1$ bei $X_2 = t_2$ und $X_1 > t_1$ bei $X_1 = t_3$ partitioniert. Abschließend wird die Region $X_1 > t_3$ bei $X_2 = t_4$ abgegrenzt. Das Ergebnis des Prozesses ist eine Partitionierung des Merkmalsraums in fünf Regionen R_1, R_2, R_3, R_4, R_5. Das entsprechende Regressionsmodell (2.5.3) berechnet Y mit einem konstanten c_m für jede Region R_m, d. h.

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}. \tag{2.5.3}$$

Das selbe Modell kann durch einen Binärbaum dargestellt werden (siehe linker unterer Teilausschnitt der Abbildung 24). Der anfängliche Datensatz befindet sich wie beschrieben im Wurzelknoten. Alle Elemente, die die Bedingungen an einem Knotenpunkt erfüllen,

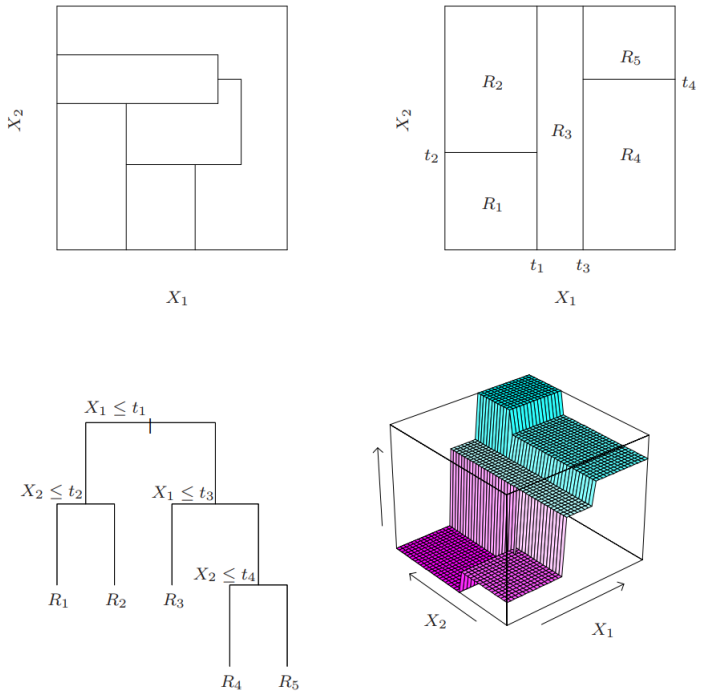


Abbildung 24: Aufteilung des Merkmalsraums

werden dem linken Ast und die anderen dem rechten Ast zugeordnet. Die Terminalknoten des Baumes entsprechen den Regionen $R_1, \ldots, R_5$. Der untere rechte Teilausschnitt der Abbildung 24 stellt ein perspektivisches Diagramm der Regressionsfläche des Modells dar.

Ein wesentlicher Vorteil des Binärbaums ist seine Interpretierbarkeit. Der Merkmalsraum wird durch einen einzigen Baum vollständig beschrieben. Bei mehr als zwei Eingaben sind Partitionen, wie im rechten oberen Teilausschnitt der Abbildung 24 veranschaulicht, schwer bildlich darzustellen.

In welcher Form sollte ein Regressionsbaum nun erweitert werden? Die Daten bestehen aus p Eingabeelementen und einer Ausgabe, für jede der N Beobachtungen, d. h. (x_i, y_i) für $i = 1, 2, \ldots, N$, mit $x_i = (x_{i1}, x_{i2}, x_{ip})$. Der Algorithmus muss automatisch entscheiden, welche Aufteilungsvariablen, Aufteilungspunkte und welche Topologie der Baum erhalten soll. Angenommen, es läge eine Aufteilung in M Regionen R_1, R_2, R_3, R_4, R_5 vor und man modelliert den Rückgabewert als eine Konstante c_m für jede Region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{2.5.4}$$

Falls man die Minimierung der Summe der Quadrate $\sum(y_i - f(x))^2$ als das Kriterium ansieht, ist leicht zu erkennen, dass der beste Wert $\hat{c}_m$ genau der Durchschnitt von y_i in der Region R_m ist (siehe (2.5.5)):

$$\hat{c}_m = \text{avg}(y_i \mid x_i \in R_m). \tag{2.5.5}$$

Die nun beste binäre Partition in Bezug auf die minimale Summe der Quadrate zu finden, ist im Allgemeinen rechnerisch unlösbar, daher wird ein Greedy-Algorithmus eingesetzt. Greedy-Algorithmen wählen einen möglichen Folgezustand aus, der zum Zeitpunkt der Wahl den größten Gewinn beziehungsweise das beste Ergebnis ermöglicht [27]. Man beginnt mit allen Daten, betrachtet eine Aufteilungsvariable j und einen Aufteilungspunkt s und definiert das Paar an Halbebenen,

$$R_1(j,s) = \{X \mid X_j \leqslant s\} \text{ und } R_2(j,s) = \{X \mid X_j > s\}. \tag{2.5.6}$$

Anschließend wird die Aufteilungsvariable j und der Aufteilungspunkt s ermittelt und man löst die folgende Gleichung:

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i - c_2)^2\right]. \tag{2.5.7}$$

Für jede Wahl von j und s wird die innere Minimierung durch:

$$\hat{c}_1 = \text{avg}(y_i \mid x_i \in R_1(j,s)) \text{ und } \text{avg}(y_i \mid x_i \in R_2(j,s)) \tag{2.5.8}$$

gelöst.

Für jede Aufteilungsvariable j kann die Bestimmung des Aufteilungspunkts s sehr schnell erfolgen. Somit kann durch das Absuchen aller Eingaben die Bestimmung des optimalen Paares (j,s) ermöglicht werden. Nachdem diese Aufteilung gefunden wurde, partitioniert man die Daten in die beiden resultierenden Bereiche und wiederholt den Aufteilungsprozess für jede der beiden Regionen. Anschließend wird dieser Prozess auf allen resultierenden Bereichen wiederholt.

Wie groß sollte der Baum nun werden? Offensichtlich könnte ein sehr großer Baum die Daten overfitten, während ein kleiner Baum die wichtige Struktur nicht erfassen kann. Die Baumgröße ist ein Tuningparameter, der die Komplexität des Modells bestimmt. Die optimale Baumgröße sollte adaptiv an den Daten bestimmt werden.

Ein Ansatz wäre es, die Aufteilung der Baumknoten nur dann vorzunehmen, falls der durch die Aufteilung bedingte Rückgang der Quadratsumme einen bestimmten Schwellenwert überschreitet. Eine bevorzugte Strategie kann es sein, einen großen Baum T_0 zu erzeugen, wobei der Aufteilungsprozess nur dann stoppt, falls eine minimale Knotengröße erreicht wurde. Dieser Baum wird unter Zuhilfenahme eines cost-complexity prunings (dt. Kostenkomplexitätsschnitt), der nachfolgend beschrieben wird, beschnitten. Sei ein Subbaum $T \subset T_0$ definiert als ein Baum, der durch das Beschneiden von T_0 erzeugt werden kann, d. h. durch Verschmelzen beliebiger interner Knoten (keine Blattknoten). Terminalknoten werden hierbei mit m indexiert, wobei m die Region R_m beschreibt. Sei $|T|$ die Anzahl an Terminalknoten in T und sei

$$\begin{aligned} N_m &= \#\{x_i \in R\}, \\ \hat{m} &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \end{aligned} \tag{2.5.9}$$

damit wird das cost complexity criterium (dt. Kriterium der Kostenkomplexität) wie folgt definiert:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|. \tag{2.5.10}$$

Die Idee ist es nun, für jedes α den Teilbaum $T_\alpha \subseteq T_0$ zu finden, um $C_\alpha(T)$ zu minimieren. Der Tuningparameter $\alpha \geqslant 0$ regelt den Kompromiss zwischen der Baumgröße und der Anpassungsfähigkeit an die bestehenden Daten. Ein großer α-Wert führt zu kleineren Bäumen T_α und umgekehrt, d. h. für ein $\alpha = 0$ erhält man den vollständigen Baum T_0. Für jedes α kann gezeigt werden, dass es einen eindeutigen kleinsten Teilbaum T_α gibt, der $C_\alpha(T)$ minimiert. Um T_α zu erhalten, wird der Ansatz weakest link pruning (dt. Entfernung des schwächstes Glieds) verwendet. Man verschmilzt nacheinander einen internen Knoten, der den geringsten Anstieg in $\sum_m N_m Q_m(T)$ erzeugt und fährt dann solange fort, bis man den Wurzelknoten erhalten hat. Dies erzeugt eine (endliche) Sequenz an Teilbäumen und man kann zeigen, dass hierbei T_α enthalten sein muss. Für tiefer gehende Details sei auf Breiman et al. [17] und Ripley [108] verwiesen. Die Schätzung von α erfolgt durch das fünffache oder zehnfache der Kreuzvalidierung. Der Wert $\hat{\alpha}$ minimiert hier die kreuzvalidierte Summe der Quadrate. Sei $T_{\hat{\alpha}}$ der finale Baum.

2.5.3.3 *Klassifizierungsbäume*

Falls es nun das Ziel ist, Werte von $1, 2, \ldots, K$ zu klassifizieren, muss nur eine einzige Änderungen im Algorithmus vorgenommen werden. Das betrifft das Aufteilen der Knoten und das Beschneiden des Baums. In einem Knoten m, der eine Region R_m mit N_m Beobachtungen repräsentiert, sei

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k), \tag{2.5.11}$$

der Anteil an Beobachtungen der Klasse k in Knoten m. Man klassifiziert die Beobachtungen in Knoten m zur Klasse $k(m) = \arg\max_k \hat{p}_{mk}$, als die Mehrheitsklasse in Knoten m. Es existieren unterschiedliche Messungen für die node impurity $Q_m(T)$ (dt. Knotenunreinheit):

$$\begin{aligned}
\text{Klassifizierungsfehler:} &\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)} \\
\text{Gini-Index:} &\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \\
\text{Kreuzentropie oder Abweichung:} &- \sum_{k=1}^{K} \hat{p}_{mk} \log\ \hat{p}_{mk}
\end{aligned} \tag{2.5.12}$$

Für zwei Klassen, falls p der Anteil der zweiten Klasse ist, sind die jeweils drei entsprechenden Messgrößen $1 - \max(p, 1-p)$, $2p(1-p)$ und $-p \log\ p - (1-p) \log\ (1-p)$. Alle drei sind ähnlich, aber Kreuzentropie und der Gini-Index [57] sind differenzierbar und somit besser zur Optimierung geeignet. Darüber hinaus sind die Kreuzentropie und der Gini-Index empfindlicher auf Veränderungen der Knotenwahrscheinlichkeiten als die Klassifizierungsfehlerrate. Aus diesem Grund sollte beim Erweitern des Baumes entweder der Gini-Index oder die Kreuzentropie verwendet werden. Um das cost-complexity pruning zu steuern, kann jeder der drei genannten Ansätze verwendet werden, aber typischerweise ist es die Klassifizierungsfehlerrate, die genutzt wird.

2.5.3.4 *Bagging und Boosting*

Die Kombination von Entscheidungen unterschiedlicher Modelle bedeutet die Zusammenführung von unterschiedlichen Ergebnissen zu einer einzigen Vorhersage. Beim Bagging (von engl. Bootstrap aggregating) sind aufeinanderfolgende Bäume nicht von Bäumen aus vorherigen Iterationen abhängig. Sie werden jeweils unabhängig voneinander mittels Bootstrap-Samples aus einem gegebenen Datensatz aufgebaut. Das Bagging kombiniert mehrere Vorhersagen gleichgewichtet miteinander und gibt am Ende den Durchschnitt der Vorhersagen

aus. Im Falle der Klassifizierung kann das Bagging im einfachsten Falle z. B. ein Mehrheitsentscheid sein, d. h. ist ein Element zwei mal der Klasse k und drei Mal der Klasse $k+1$ zugeordnet worden, wird das Element der letzteren zugewiesen. Im Falle der numerischen Vorhersage ist es z. B. der Durchschnitt, den es zu berechnen gilt.

Beim Bagging wird angenommen, dass es mehrere Trainingsdatensätze gleicher Größe gibt, die zufällig aus einer Instanz ausgewählt werden. Man stelle sich vor, man verwende einen bestimmten Ansatz aus dem maschinellen Lernen, um einen Entscheidungsbaum für jeden Datensatz zu erstellen. Man könnte erwarten, dass diese Bäume de facto identisch sind und man für jede neue Testinstanz identische Vorhersagen erhält. Überraschenderweise ist diese Annahme in der Regel leider falsch, insbesondere dann, wenn die Trainingsdatensätze relativ klein sind. Leichte Änderungen der Trainingsdaten können schnell dazu führen, dass an einem bestimmten Knoten ein anderes Attribut ausgewählt wird und zwar mit erheblichen Auswirkungen auf die Struktur des jeweiligen Teilbaums. Dies bedeutet, dass es Testinstanzen gibt, für die einige der Entscheidungsbäume korrekte Vorhersagen liefern und für andere nicht [124].

Bäume können miteinander kombiniert werden, indem man sie für jede Testinstanz eine Entscheidung treffen lässt. Wird ein bestimmtes Element öfters einer gewissen Klasse in einer Testinstanz zugeordnet als jedes andere Element, wird es dieser Klasse zugewiesen. Generell gilt, je mehr Stimmen berücksichtigt werden, desto zuverlässiger werden die Vorhersagen für jede Klasse [124]. Insbesondere wird der kombinierte Klassifikator selten ungenauer sein als ein einzelner Entscheidungsbaum, der nur aus einem der Datensätze besteht.

Sowohl Bagging als auch Boosting (dt. Verstärken) verfolgen diesen Ansatz. Sie leiten diese einzelnen Modelle aber auf unterschiedliche Weise ab. Das Boosting verbindet viele schwache Algorithmen zu einem mächtigen Klassifikator. Die Kombination mehrerer Modelle hilft aber nur dann, wenn sich diese Modelle auch deutlich voneinander unterscheiden und wenn jedes einen angemessenen Prozentsatz der zu untersuchenden Daten korrekt klassifiziert. Im Idealfall ergänzen sich die Modelle gegenseitig. Beim Boosting erhalten nachfolgende Bäume ein zusätzliches Gewicht, falls durch die vorherigen Prädiktoren falsche Vorhersagen getroffen wurden. Es existieren viele unterschiedliche Ansätze. Die Gewichtung beim Boosting wird dazu verwendet, um den potenziell erfolgreicheren Elementen mehr Bedeutung bei der Berechnung des Ergebnisses zuzuweisen. Der wohl bekannteste Boosting Algorithmus ist AdaBoost [89, 124].

3

VERWANDTE THEMEN ZUR SUBSTRUKTURSUCHE

Erste Algorithmen zur Substruktursuche wurden bereits in den 1960er und 1970er Jahren von Cossum [28] und Figueras [52] entwickelt. Dieses Kapitel gibt einen Überblick über die bekanntesten Arbeiten zur Substruktursuche. Die Substruktursuche ist ein $\mathcal{NP}$-vollständiges Problem. Für diese Klasse an Problemen sind keine Algorithmen mit polynomieller Laufzeit bekannt, wenn $\mathcal{P} \neq \mathcal{NP}$ gilt. Die bekanntesten Arbeiten zur Substruktursuche sind die von Ullmann [117] und Cordella et al. [26]. In dem Abschnitt 3.1 wird der Algorithmus von Ullmann, in Abschnitt 3.2 der VF2-Algorithmus von Cordella et al. vorgestellt.

3.1 ALGORITHMUS VON ULLMANN

Der Algorithmus von Ullmann [117] ist ein Backtracking-Algorithmus zur Substruktursuche. Der Algorithmus formuliert das Subgraph-Isomorphie-Problem unter Zuhilfenahme von Matrizen. Seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ zwei Graphen mit jeweils einer Adjazenzmatrix $A = [a_{i,j}]$ und $B = [b_{i,j}]$ und seien $n_1 = |V_1|$, $n_2 = |V_2|$. Seien M^0, M' und $C = [c_{i,j}] = M'(M'B)^T$ Matrizen. Eine Adjazenzmatrix eines Graphen speichert in einer $n \times n$ Matrix, welche Knoten durch eine Kante miteinander verbunden sind. Ein Eintrag (i, j) gibt hierbei an, ob eine Kante zwischen den Knoten i und j besteht, wobei i die Spalten und j die Zeilen einer Matrix darstellen. Weist der Eintrag einen Wert 1 auf, besteht eine Kante, bei einem Wert von 0 keine [12]. Ein Subgraph-Isomorphismus ist ein Graph-Isomorphismus zwischen einem Graphen G_2 und einem Subgraphen G_1.

Der Algorithmus geht bei der Berechnung des Subgraph-Isomorphismus wie folgt vor: Am Anfang erstellt der Algorithmus eine $n_1 \times n_2$ Matrix $M^0 = [m^0_{i,j}]$. Ein Element in $m_{i,j} \in M'$ wird hierbei auf 1 gesetzt, falls der Grad des j-ten Knotens in G_2 größer oder gleich dem Grad des i-ten Knotens in G_1 ist, andernfalls wird der Wert auf 0 gesetzt. Siehe folgendes Beispiel [90]:

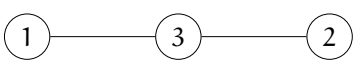


Abbildung 25: Ullmann – Graph G_1

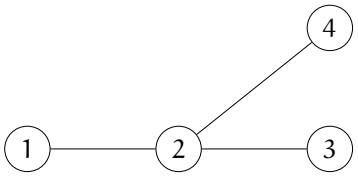


Abbildung 26: Ullmann – Graph G_2

Sei in Abbildung 25 der Graph G_1 und in Abbildung 26 der Graph G_2 dargestellt. Seien A die Adjazenzmatrix des Graphen G_1, B des Graphen G_2 und M^0 die resultierende Matrix (siehe Matrizen aus (3.1.1)).

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \; B = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \; M^0 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \tag{3.1.1}$$

Der Algorithmus versucht nun, iterativ Einsen aus der initialen Matrix M^0 zu entfernen und alle möglichen Matrizen M' aus M^0 zu erzeugen. Dabei durchläuft er per DFS-Aufruf den Suchbaum bis eine Matrix M' erzeugt wurde, die die Bedingung aus (3.1.2) erfüllt. Dabei darf keine Spalte in der Matrix M' mehr als eine 1 und jede Zeile sollte mindestens eine 1 aufweisen. Der Algorithmus versucht, iterativ die Einträge einer Zeile alle bis auf eine 1 auf 0 zu setzen. Ist eine gültige Matrix M' erzeugt worden, wird mit ihr die Matrix C berechnet und mit der Adjazentmatrix von A verglichen. Entspricht die Matrix C der Matrix A, ist ein isomorpher Subgraph gefunden worden.

$$\forall\, i, j \in \{0, \ldots, |V_1| - 1\} : (a_{i,j} = 1) \Leftrightarrow (c_{i,j} = 1) \tag{3.1.2}$$

Beispiel: Am Anfang werden alle Einsen aus der Matrix M^0 in Zeile eins, d. h. alle Einträge aus $m^0_{1,j}$, außer dem Eintrag $m^0_{1,1}$ auf 0 gesetzt. Die resultierende Matrix ist in (3.1.3) dargestellt. Nach jeder Operation wird die Bedingung aus (3.1.2) geprüft. Falls die Bedingung den Wert `false` ausgibt, fährt man fort.

$$M'^0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \tag{3.1.3}$$

Mittels Tiefensuche versucht der Algorithmus weitere Zeilen mit nur einer 1 zu erzeugen. Da jede Spalte maximal eine 1 aufweisen darf, wird in der Zeile $m'^{0}_{2,j}$ der Eintrag $m'^{0}_{2,2}$ auf 1 und der Rest der Zeile auf 0 gesetzt. Daraus folgt die Matrix (3.1.4).

$$M'^{1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{3.1.4}$$

Nun geht der Algorithmus wieder eine Zeile tiefer und bearbeitet Zeile $m'^{1}_{3,j}$. Da die Zeile $m'^{1}_{3,j}$ nur genau eine 1 in Eintrag $m'^{1}_{3,2}$ aufweist, aber bereits in Eintrag $m'^{1}_{2,2}$ eine 1 notiert ist, muss ein Backtrackingschritt erfolgen, weil jede Spalte maximal eine 1 aufweisen darf. Da kein anderer Eintrag in Zeile $m'^{1}_{3,j}$ mit einer 1 ausgezeichnet werden kann (weil keiner vorhanden ist), muss ein anderer Eintrag in $m'^{1}_{2,j}$ mit einer 1 ausgezeichnet werden (siehe Matrix (3.1.5)). Dabei wird der alte Zustand aus M^0 in der jeweiligen Zeile wiederhergestellt und eine andere 1 ausgezeichnet, falls eine vorhanden war.

$$M'^{2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{3.1.5}$$

Nun erfolgt, wie erläutert, ein Backtrackingschritt und der Algorithmus springt zur Matrix (3.1.4). Dabei wird der Eintrag $m'^{1}_{2,2}$ auf 0 und ein anderer Eintrag, hier der Eintrag $m'^{1}_{2,3}$, auf 1 geändert (siehe Matrix (3.1.6)). Es kann nur ein Eintrag in der Zeile auf 1 gesetzt werden, falls der jeweilige Eintrag in der Matrix M^0 auch eine 1 aufwies. Im Anschluss wird die Bedingung (3.1.2) erneut geprüft.

$$M'^{3} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{3.1.6}$$

Nun steigt der Algorithmus wieder eine Zeile zu $m'^{3}_{3,j}$ ab. Da nur eine 1 in dieser Zeile vorhanden ist, bleibt im Eintrag $m'^{3}_{3,2}$ die 1 bestehen und die Bedingung aus (3.1.2) wird erneut geprüft. Gilt die Bedingung aus (3.1.2), d. h. ist die Matrix C_1 identisch mit der Matrix A, hat der Algorithmus einen isomorphen Subgraphen gefunden.

$$M'^{4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{3.1.7}$$

Wie in Matrix (3.1.8) zu sehen ist, gleicht die Matrix C_1 der Matrix A und es wurde ein Subgraph-Isomorphismus gefunden. Somit bricht der Algorithmus die Berechnung ab. Ist dies nicht der Fall, erfolgt wieder so lange ein Backtrackingschritt bis wieder eine gültige Matrix M' berechnet werden konnte oder alle Möglichkeiten ohne Erfolg geprüft wurden.

$$C_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = A \tag{3.1.8}$$

Um den Rechenaufwand im Algorithmus von Ullmann für das Auffinden des Subgraph-Isomorphimus zu minimieren, wurde von Ullmann ein Verbesserungsansatz präsentiert, der vorab einige Einsen aus den Matrizen M' zu eliminieren versucht. Durch diesen Verbesserungsansatz werden einige Nachfolgeknoten im Suchbaum entfernt und es kann umgehend ein Backtrackingschritt erfolgen. Hierbei werden die Zeilen der Matrix nach absteigendem Knotengrad bearbeitet. Sei M die Matrix, die mit einem gegebenen nichtterminalen Knoten im Suchbaum verknüpft ist. Jeder Subgraph-Isomorphismus entspricht genau einer bestimmten Matrix M'. Ein Isomorphismus ist ein Isomorphismus unter M, falls sein Endknoten im Suchbaum ein Nachfolger des Knotens ist, dem M zugeordnet ist. Die Nullen in der Matrix M verhindern lediglich Übereinstimmungen zwischen den Knoten aus V_a und V_b. Falls für alle Isomorphismen unter M $m'_{i,j} = 0$ gilt und falls dann $m_{i,j} = 1$ ist, kann $m_{i,j} = 1$ zu $m_{i,j} = 0$ geändert werden, ohne einen Isomorphismus in M einzubüßen - diese Isomorphismen werden weiterhin durch den DFS-Aufruf gefunden. Sei v_{a_i} der i-te Eintrag in V_a und v_{b_j} der j-te Eintrag in V_b. Sei $\{v_{a_i}, \ldots, v_{a_x}, \ldots, v_{a_\gamma}\}$ die Menge aller Einträge in G_a, die adjazent zu v_{a_i} in G_a sind. Betrachtet man die Matrix M', die mit jedem Isomorphismus in M in Zusammenhang steht, d. h. falls v_{a_i} mit v_{b_j} in jedem Isomorphismus in M übereinstimmt, gilt für alle $x \in \{0, \ldots, |V_1| - 1\}$:

$$(a_{i,x} = 1) \Rightarrow \exists y \in \{0, \ldots, |V_2| - 1\} : (m_{x,y} b_{y,i} = 1) \tag{3.1.9}$$

Dieser Verbesserungsansatz testet, dass jede 1 in M die Bedingung aus (3.1.9) erfüllt. Für jedes $m_{i,j} = 1$, das die Bedingung aus (3.1.9) nicht erfüllt, wird der Wert von 1 auf 0 verändert.

Bei der Implementierung des Algorithmus muss gewährleistet werden, dass ein Zustand nach einem Backtrackingschritt wiederhergestellt werden kann. Ein Suchpattern mit n Knoten und einem Graphen mit m Knoten weist eine WorstCase-Laufzeit von $\mathcal{O}(m^n n^2)$ auf. Der Speicherbedarf kann durch $\mathcal{O}(m^3)$ abgeschätzt werden [26]. Der

Algorithmus hat sich in Evaluationen bereits als geeignet erwiesen [127]. Im Falle einer Eins-zu-Eins-Übereinstimmung (d. h. zwei Graphen sind isomorph) zwischen zwei Graphen ist der Algorithmus von Ullmann besser geeignet als andere [53].

3.2 VF2-ALGORITHMUS VON CORDELLA ET AL.

Ebenso wie der Algorithmus von Ullmann [117] basiert der Ansatz von Cordella et al. [26] auf einem Backtrackingalgorithmus, der iterativ versucht, eine partielle Abbildung zu erweitern. Ein gegebenes Regelwerk ermöglicht es hierbei den Suchraum zu beschränken. Der VF2-Algorithmus von Cordella et al. wurde 1990 entwickelt.

Seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ zwei Graphen mit den dazugehörigen Knoten- und Kantenmengen. Ein Matching M zwischen zwei Graphen G_1 und G_2 stellt hierbei eine partielle Abbildung dar. Die Knoten von G_1 werden mit Knoten von G_2 durch ein gegebenes Regelwerk verknüpft. Seien $n \in V_1$ und $m \in V_2$. Ein Mapping M wird als eine Menge an Paaren $p = (n, m)$ dargestellt. Jedes Paar repräsentiert ein Mapping eines Knotens n auf einen Knoten m. Ein Mapping $M \subset V_1 \times V_2$ ist ein Isomorphismus, falls M eine bijektive Funktion ist, die die Kantenstruktur zweier Graphen G_1 und G_2 erhält. Ein Mapping M ist ein Subgraph-Isomorphismus zwischen den Graphen G_1 und G_2, falls M ein Isomorphismus zwischen G_2 und einem Subgraphen von G_1 ist. Jeder Zustand s des Matching-Prozesses kann einer partiellen Mapping-Lösung $M(s)$ zugeordnet werden. Eine Mapping-Lösung $M(s)$ enthält nur eine Teilmenge von M .

Im Folgenden seien $M_1(s), M_2(s), B_1(s)$ und $B_2(s)$ die Mengen an Knoten und Kanten von $G_1(s)$ und $G_2(s)$. $M(s)$ identifiziert eindeutig zwei Subgraphen von G_1 und G_2. $G_1(s)$ und $G_2(s)$ werden erzeugt, indem Knoten und Kanten aus G_1 und G_2 ausgewählt werden, die in $M(s)$ enthalten sind. Eine Transition eines generischen Zustandes s zu einem Nachfolger s' wird durch das Hinzufügen eines Paares (n, m) an übereinstimmenden Knoten zu einem Teilgraphen erzeugt. Es kann gezeigt werden, dass im Falle von Graph-Isomorphie oder Subgraph-Isomorphie, die partiellen Graphen $G_1(s)$ und $G_2(s)$ zu $M(s)$ gehören und isomorph sind. Der VF2-Algorithmus definiert Regeln, die die Konsistenzbedingungen verifizieren. Somit werden nur konsistente Zustände erzeugt. Die Anzahl erzeugter Zustände kann ferner dadurch reduziert werden, indem eine Menge an Regeln, die von Cordella et al. als k-Look-Ahead-Regeln (dt. k-Vorausschauende Regeln) bezeichnet, eingeführt werden. Diese Regeln werden zur Vorprüfung eingeführt, um zu ermitteln, ob ein konsistenter Zustand s keine konsistenten Nachfolger s' nach k Schritten aufweist. Die k-Look-Ahead-Regeln werden *feasibility rules* (dt. Durchführbarkeitsregeln, DFF) genannt. Eine Durchführbarkeitsfunktion $F(s, n, m)$ ist

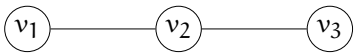


Abbildung 27: VF2 – Graph G_1

true, falls das Hinzufügen eines Paares $p = (n, m)$ zu s die DFF erfüllt. Sie hängen nur von der Struktur des Eingangsgraphen ab. Falls die Eingabegraphen jedoch Knoten- und Kantenlabel aufweisen, müssen sie ebenfalls berücksichtigt werden, daher ist die DFF wie folgt definiert:

$$F(s, n, m) = F_{syn}(s, n, m) \wedge F_{sem}(s, n, m).$$

$F_{syn}(s, n, m)$ hängt von der Struktur des Graphen und $F_{sem}(s, n, m)$ von seinen Attributen ab.

Am Anfang besteht das partielle Matching $M(s_0) = \emptyset$, wobei s_0 den initialen Zustand eines Matchings M darstellt, das keine Komponenten aufweist. Für jeden Zwischenzustand s berechnet der Algorithmus eine Menge $P(s)$ an Knotenpaaren $p = (n, m)$, die zu dem aktuellen Zustand s hinzugefügt werden sollen. Jedes Paar p, das zu $P(s)$ gehört, muss den Matching-Bedingung genügen, d. h. $F(s, n, m)$ wird in Folge dessen zu true ausgewertet. Ein Nachfolgezustand $s' = s \cup p$ wird berechnet und der ganze Prozess wird rekursiv auf s' angewendet. Damit der Algorithmus während des Matching-Prozesses keine unbrauchbaren und bereits generierten Zustände berechnet, wird ein spezielles Verfahren zur Erzeugung eines Nachfolgerknotens verwendet. Da die Reihenfolge der Knoteneinfügung zur Teillösung $M(s)$ den resultierenden Zustand s nicht beeinflusst, ignoriert der Algorithmus jedes Paar (n_i, m_j) in $P(s)$, falls die Menge $M(s)$ bereits die Knoten aus (n_i, m_j) enthält. Somit wird sichergestellt, dass jeder Zustand nur einmal erzeugt wird. Die Menge aller möglichen Paare $P(s)$, die zu dem aktuellen Zustand s hinzugefügt werden können, sind die Knotenmengen, die direkt mit $G_1(s)$ und $G_2(s)$ verbunden sind. Seien $T_1^{out}(s) \subset V_1$ und $T_2^{out}(s) \subset V_2$ die Knotenmengen, die noch nicht im partiellen Matching $M(s)$ enthalten sind, aber Ziel einer *ausgehenden Kante* aus $G_1(s)$ beziehungsweise $G_2(s)$ sind. Seien $T_1^{in}(s) \subset V_1$ und $T_2^{in}(s) \subset V_2$ die Knotenmengen, die noch nicht im partiellen Matching $M(s)$ enthalten sind, aber Ursprung einer *eingehenden Kante* in $G_1(s)$ beziehungsweise $G_2(s)$ darstellen. Die Menge $P(s)$ wird aus allen Knotenpaaren $p = (n, m)$ gebildet, wobei n Element von $T_1^{out}(s)$ und m Element von $T_2^{out}(s)$ ist, sofern eine dieser beiden Mengen nicht leer ist. Sollten diese Mengen leer sein, wird die Menge $P(s)$ durch $T_1^{in}(s)$ bzw. $T_2^{in}(s)$ erzeugt. Im Falle von nicht verbundenen Graphen können für einige Zustände s

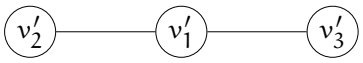


Abbildung 28: VF2 – Graph G_2

die obigen Mengen leer sein. In diesem Fall ist die Menge an Kandidatenpaaren, die $P(s)$ bilden können, die Menge P_s^d, die weder aus Knoten aus der Knotenmenge G_1 noch aus der Knotenmenge von G_2 besteht, d. h. $P(s)$ setzt sich aus Knoten zusammen, die nicht in $M(s)$ enthalten sind. Die Menge an Knoten, die nicht direkt mit der partiellen Abbildung $M(s)$ verbunden sind, werden mit $V_1(s)$ und $V_2(s)$ bezeichnet.

Cordella et al. definieren fünf Durchführbarkeitsregeln, die mittels k-Look-Ahead-Regeln prüfen, welche Folgezustände (s) ausgeschlossen werden können, deren partielle Abbildung $M'(s)$ keine konsistenten Zustände aufweisen, um p zu $M'(s)$ hinzufügen zu können. Diese sind: $\mathcal{R}_{pred}$, $\mathcal{R}_{succ}$, $\mathcal{R}_{in}$, $\mathcal{R}_{out}$ und $\mathcal{R}_{new}$. Die ersten beiden Regeln $\mathcal{R}_{pred}$ und $\mathcal{R}_{succ}$ testen die Konsistenz einer partiellen Lösung $M(s')$, d. h. durch das Hinzufügen des betrachteten Kandidatenpaars (n, m) zur aktuellen partiellen Lösung $M(s)$ wird $M(s')$ erzeugt. Die restlichen Regeln schneiden Möglichkeiten aus dem Suchbaum heraus. Insbesondere führen $\mathcal{R}_{in}$ und $\mathcal{R}_{out}$ eine 1-Look-Ahead und $\mathcal{R}_{new}$ einen 2-Look-Ahead im Suchprozess aus.

Sei in Abbildung 27 der Graph $G_1 = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3)\}$ und in 28 der Graph $G_2 = (\{v_1', v_2', v_3'\}, \{(v_1', v_3'), (v_1', v_2')\}$ gegeben. Um das Vorgehen des vorgestellten Algorithmus zu verdeutlichen, wird nachfolgend getestet, ob der Graph G_1 isomorph zu G_2 ist. Der VF2 Algorithmus geht wie Folgt vor:

Iteration 1:
Der Algorithmus besteht am Anfang aus dem leeren Matching s_0.

Iteration 2:
Der Knoten v_1 wird mit dem Knoten v_1' gematcht.

Iteration 3:
Der Knoten v_2 wird mit dem Knoten v_2' gematcht.

Iteration 4:
Der Knoten v_3 kann aber nicht mit dem Knoten v_3' gematcht werden, da keine Kante zwischen Knoten v_3' und v_2' aber eine zwischen den Knoten v_1 und v_3 besteht. Backtrack zu Iteration 2.

Iteration 5:
Der Knoten v_2 wird mit dem Knoten v_3' gematcht.

Iteration 6:
Der Knoten v_3 kann nicht mit dem Knoten v'_2 gematcht werden (Grund: siehe Iteration 4). Backtrack zu Iteration 1.

Iteration 7:
Der Knoten v_1 wird mit dem Knoten v'_2 gematcht.

Iteration 8:
Der Knoten v_2 wird mit dem Knoten v'_1 gematcht.

Iteration 9:
Der Knoten v_3 wird mit dem Knoten v'_3 gematcht. Als Ergebnis wurde der Knoten v_1 auf v'_2, der Knoten v_2 auf v'_1 und der Knoten v_3 auf v'_3 gematcht, d. h. es ist ein isomorpher Graph gefunden worden. Falls alle Möglichkeiten getestet, aber kein gültiges Ergebnis gefunden wurde, sind die getesteten Graphen nicht isomorph zueinander [98].

Die Laufzeit des Algorithmus wird mit $\mathcal{O}(m!m)$ und der Speicherbedarf mit $\mathcal{O}(m)$ abgeschätzt, wobei $m = |V_2| \geqslant |V_1|$ ist. Aufgrund des geringen Speicherbedarfs eignet sich der VF2-Algorithmus auch auf Graphen mit einer hohen Anzahl an Knoten und Kanten [26]. Eine Vergleichsstudie zeigte, dass der VF2 Algorithmus sowohl im Best- als auch im Worst-Case eine bessere Laufzeit aufweist und weniger Speicher benötigt als der Algorithmus von Ullmann [53].

4

DEFINITIONEN ZUM ALGORITHMUS ZUR SUBSTRUKTURSUCHE

Nachfolgend werden die formalen Definitionen für die Kapitel 5 und 6 eingeführt.

Definition 4.1 (Graph und Graphkomponenten)**.**
Sei G *ein knoten- und kantengelabelter Multigraph,* $\mathcal{L}_V^G$ *eine Knoten- und* $\mathcal{L}_E^G$ *eine Kantenlabelmenge. Ein Graph* G *über* $(\mathcal{L}_V^G, \mathcal{L}_E^G)$ *ist ein Tupel* $G = (V^G, E^G, N^G, L_V^G, L_E^G)$*. Seien* V^G *die Knotenmenge,* E^G *die Kantenmenge,* $N^G : E^G \to (V^G)^2$ *die eindeutige Zuordnung der Kanten zu ihren Knoten.* $L_V^G : V^G \to \mathcal{L}_V^G$ *weist jedem Knoten und* $L_E^G : E^G \to \mathcal{L}_E^G$ *jeder Kante ein Label zu und sei* $(V^G)^2$ *eine Teilmenge der Kardinalität 2 von* V^G*. Sei für einen Graphen* $G = (V^G, E^G, N^G, L_V^G, L_E^G)$ *und* $U, U' \subseteq V^G$ *mit* $U \cap U' = \emptyset$*, sei* $\delta_G(U) = \{e \in E^G \mid \emptyset \neq U \cap N^G(e) \neq N^G(e)\}$*,* $\delta_G(U, U') = \{e \in E^G \mid U \cap N^G(e) \neq \emptyset$ *and* $U' \cap N^G(e) \neq \emptyset\}$ *und sei* $G[U]$ *der induzierte Subgraph von* U*. Statt* $\delta_G(U)$ *wird im Folgenden* $\delta(U)$ *und* $\delta(v)$ *statt* $\delta(\{v\})$ *als Notation verwendet.* $\delta(U)$ *enthält die benachbarten Knoten von* U*, die nicht selbst in* U *enthalten sind. Sei* $\Delta(G)$ *der maximale Knotengrad im Graphen* G*. Ein Schnitt* $(U, V \setminus U)$ *ist eine Partition der Knoten des Graphen* G *in zwei disjunkten Teilmengen, bezeichnet als der Schnitt* U*. Ein Schnitt* U *ist minimal, falls kein anderer Schnitt* U' *existiert, so dass* $\delta(U')$ *eine echte Teilmenge von* $\delta(U)$ *ist. In einem zusammenhängenden Graphen ist ein Schnitt* U *minimal, falls* $G[U]$ *und* $G[V \setminus U]$ *verbunden sind.*

Definition 4.2 (Graphersetzungssystem)**.**
Sei ein Graphersetzungssystem ein Tupel (S, P)*, das wie folgt definiert ist: Sei* $S \in \mathcal{L}_V^G$ *ein Knotenlabel und* P *eine endliche Menge an Ersetzungsregeln der Form:*

$$(L_V^G, L_E^G(e_1^G), \ldots, L_E^G(e_d^G)) \to (G, n_1^G, \ldots, n_d^G),$$

wobei $L_V^G \in \mathcal{L}_V^G$*,* $L_E^G(e_1^G), \ldots, L_E^G(e_d^G) \in \mathcal{L}_E^G$*,* G *der Graph und* $n_1^G, \ldots, n_d^G$ *die Knoten des Graphen* G *sind. Sei* d *der Grad einer Ersetzungsregel und* G *der Ersetzungsgraph. Sei für ein Graphenersetzungssystem* (S, P) *der maximale Grad einer Ersetzungsregel* $\Delta(S, P)$ *und* $L_v^G = L_V^G(v)$*. Solch eine Regel erlaubt das Ersetzen eines Knotens mit dem Label* L_v^G *und dessen inzidente Kanten* $L_E^G(e_1^G), \ldots, L_E^G(e_d^G)$ *durch den Graphen* G*. Wird eine Ersetzungsregel angewendet, erhalten die inzidenten Kanten des entfernten Knotens* L_v^G *neue Endpunkte* $n_1^G, \ldots, n_d^G$ *in genau dieser Reihenfolge zugeordnet. Formeller, sei ein Graph* $H \subseteq G$*,* $H = (V^H, E^H, N^H, L_V^H, L_E^H)$ *gegeben. Eine Ersetzungsregel*

$$(L_V^G, L_E^G(e_1^G), \ldots, L_E^G(e_d^G)) \to (V^G, E^G, N^G, L_V^G, L_E^G, n_1^G, \ldots, n_d^G))$$

kann auf einen Knoten $v \in V^H$ *angewendet werden, falls eine Bijektion* $m : \{1, \ldots, d\} \to \delta_H(v)$, *so dass* $L_E(m(i)) = L_{E_i}$ *für* $1 \leqslant i \leqslant d$ *und* $L_V^H(v) = L_V$ *existiert. Falls diese Regel unter Verwendung von* m *auf* v *angewendet wird, ist* $H' = (V^{H'}, E^{H'}, N^{H'}, L_V^{H'}, L_E^{H'})$ *der resultierende Graph mit:*

- $V^{H'} = V^H \setminus \{v\} \cup V^G$. *Dabei wird angenommen, dass die Knoten* V^G *disjunkt zu den Knoten* V^H *sind (ist dies nicht der Fall, wird vorher eine Isomorphe Kopie von* G *erstellt),*
- $E^{H'} = E^H \setminus \{m(1), \ldots, m(d)\} \cup E^G \cup \{e_1, \ldots, e_d\}$, *wobei* $e_1, \ldots, e_d$ *neue Kanten darstellen,*
- $N^{H'}$ *ist zu definieren als* $N^{H'}(e) = N^H(e)$, *falls* $e \in E^{H'} \cap E^H$, $N^{H'}(e) = N^G(e)$, *falls* $e \in E^G$ *und* $N^{H'}(e_i) = \{n_i\} \cup (N^H(m(i)) \setminus \{v\})$ *für* $1 \leqslant i \leqslant d$,
- $L_V^{H'}$ *ist definiert als* $L_V^{H'}(v) = L_V^H(v)$ *für* $v \in V^{H'} \cap V^H$ *und* $L_V^{H'}(v) = L_V^G(v)$ *für* $v \in G$
- $L_E^{H'}$ *ist definiert als* $L_E^{H'}(e) = L_E^H(e)$ *für* $e \in E^{H'} \cap E^H$, $L_E^{H'}(e) = L_E^G(e)$ *für* $e \in E^G$, $L_E^{H'}(e_i) = L_E^H(m(i))$ *für* $1 \leqslant i \leqslant d$.

In den vorangegangenen Definitionen wurde bisher nicht zwischen terminalen und nichtterminalen Knotenlabeln unterschieden. Es existieren jedoch Beispiele, in denen die Knotenlabel nicht im Anfragegraphen, sondern nur auf der linken Seite der Ersetzungsregel erscheinen. Diese Knotenlabel sind Nichtterminale-Knotenlabel und werden mit einem rechteckigen Rahmen notiert. Die anderen Knotenlabel sind Terminale und weisen einen Kreis auf. Für eine Beschreibung wird auf Abbildung 29 verwiesen. Im Folgenden wird mitunter zusätzlich verlangt, dass der Graph G in einer Ersetzungsregel zusammenhängend ist.

Sei $\mathcal{G}(S, P)$ die Menge aller Graphen, die durch iterative Anwendung von Regeln erlangt werden kann, beginnend bei einem Graphen, der aus einem einzelnen Knotenlabel S und keiner Kante besteht. Sei G ein Anfragegraph und (S, P) ein Graphersetzungssystem. Die zentrale Fragestellung dieser Arbeit ist: Existiert ein Subgraph von G in $\mathcal{G}(S, P)$? Diese Abfrage weicht von den typischerweise in der Literatur gestellten Fragestellungen zu Graphersetzungssystem ab, in denen man interessiert ist, ob G in $\mathcal{G}(S, P)$ enthalten ist. Eine weitere Frage wäre die Existenz eines induzierten Subgraphen von G in $\mathcal{G}(S, P)$. Der in Kapitel 6 vorgestellte Algorithmus kann ohne erheblichen Mehraufwand auf diese Fragestellung verallgemeinert werden.

Falls der Grad eines zu ersetzenden Graphen größer als das Maximum von $\Delta(S, P)$ und $\Delta(G)$ einer Ersetzungsregel für einen Anfragegraphen G ist, kann die entsprechende Regel nicht angewendet

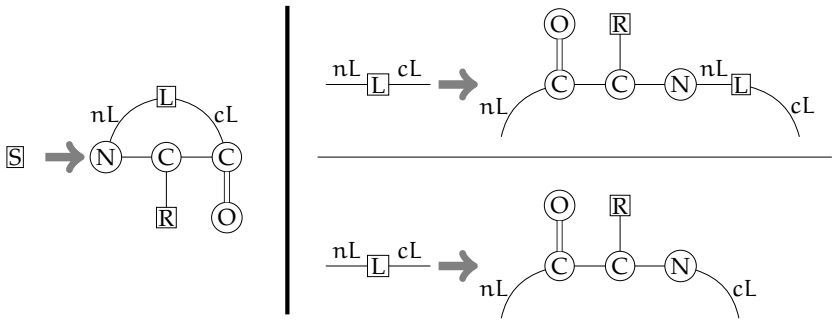


Abbildung 29: Eine Graph-Grammatik für Zyklische-Peptide

Sei S hier das Startlabel. Die linke Seite der Abbildung zeigt eine Regel, in der das Startsymbol durch einen initialen Graphen ersetzt werden kann. Sei L die Bezeichnung für eine willkürliche Kette von Aminosäuren und R die für einen einzelnen Aminosäurerest. Die rechte obere Regel erweitert die Aminosäurenkette, um eine weitere Aminosäure. Nach Anwendung der rechten unteren Regel kann eine Aminosäurenkette nicht mehr vergrößert werden. Kantenlabel werden in der obigen Abbildung dazu verwendet, um sicherzustellen, dass z. B. ein Stickstoff-Atom (N-Knoten) immer mit einem Kohlenstoff-Atom (C-Knoten) und nicht mit einem anderen Stickstoff-Atom verbunden wird. Eine Kante hat das Label nL, falls sie zwischen einem Stickstoff- und einem Knotenlabel L liegt. Eine Kante weist das Label cL auf, falls sie zwischen einem Stickstoff- und einem Knotenlabel L liegt. Inwiefern R durch eine Ersetzungsregel zu einer Aminosäure oder einem einzelnen Knoten abgeleitet wird, hängt davon ab, ob ein Peptid beschrieben wird oder nicht. Kantenlabel werden verwendet, um die Anwendbarkeit der Ersetzungsregeln zusätzlich einzuschränken. Somit kann eine höhere Ausdruckskraft erzielt werden. Eingabegraphen haben typischerweise nur Knotenlabel. Liegt ein Graph G ohne Kantenlabel vor, ist man interessiert, ob ein Kantenlabelling existiert, so dass der resultierende Graph einen Subgraphen beinhaltet, der in $\mathcal{G}(S, P)$ enthalten ist. Auch in diesem Fall kann der Algorithmus in Kapitel 6 leicht auf diese Fragestellung erweitert werden. Die Komplexität von Algorithmen zum Beschreiben von Graphersetzungssystemen wird üblicherweise unter der Annahme untersucht, dass das Graphersetzungssystem unveränderbar ist und daher ein polynomieller Algorithmus exponentiell von der Beschreibungsgröße des Systems abhängen kann. Das hier vorgestellte Graphersetzungssystem ist hingegen Teil der Eingabe.

werden, um einen Subgraphen von G abzuleiten, d. h. falls ein Knoten des zu ersetzenden Graphen einen Knotengrad größer als dieses Maximum aufweist, kann dieser Knoten weder Teil des Anfragegraphen sein noch durch eine Regel des Graphersetzungssystems ersetzt werden. Es wird daher im Folgenden angenommen, dass der Grad eines jeden Ersetzungsgraphen $\max(\Delta(S,P),\Delta(G))$ ist, falls der Algorithmus auf den Anfragegraphen G angewendet wird. Sei $\zeta(G)$ die maximale Größe eines minimalen Schnittes von G. Genauer, sei $\zeta(G)$ die maximale Kardinalität eines Schnittes $U \subseteq V$, so dass $G[U]$ und $G[V \setminus U]$ verbunden sind. Für ein Graphersetzungssystem (S,P) sei $\zeta(S,P)$ der maximale Wert $\zeta(G)$ für einen Ersetzungsgraphen G.

5

REGELN IN NORMALFORM

Um die Beschreibung des Algorithmus zu vereinfachen, wird nachfolgend erläutert, wie Ersetzungsregeln normalisiert werden. Dabei wird angenommen, dass der Ersetzungsgraph aus exakt zwei Knoten und einer beliebigen Anzahl an Kanten zwischen diesen besteht. Falls alle Ersetzungsgraphen diese Eigenschaft aufweisen, werden die Regeln in Normalform aufgerufen. Es wird aufgezeigt, dass allgemeine Ersetzungsregeln, die in Kapitel 6 definiert wurden, immer auf eine Menge an Ersetzungsregeln in Normalform reduziert werden können und dass der maximale Grad einer Ersetzungsregel eines Ersetzungsgraphen G sich um höchstens $\zeta(G) \cdot \Delta(G)$ erhöhen kann. $\Delta(G)$ kann durch den maximalen Grad der Ersetzungsregeln und des Anfragegraphen beschränkt werden. Dies erhöht den Grad einer Ersetzungsregel um maximal $\zeta(G) \cdot \Delta(G)$. Hierbei kann $\Delta(G)$ durch den maximalen Grad einer Ersetzungsregel und des Anfragegraphen beschränkt werden.

Falls der resultierende Graph nur aus einem Knoten besteht, werden keine Ersetzungsregeln benötigt. Solche Ersetzungsregeln würden lediglich das Knotenlabel verändern. Aus diesem Grund können diese Ersetzungsregeln, wie bei der Konstruktion von kontextfreien Grammatiken der Chomsky-Normalform [110], weggelassen werden.

Sei $(L_V^G, L_E^G(e_1^G), \ldots, L_E^G(e_d^G)) \rightarrow (V^G, E^G, N^G, L_V^G, L_E^G, n_1, \ldots, n_d))$ eine Ersetzungsregel in allgemeiner Form, $V^G = \{v_1^G, \ldots, v_n^G\}$, $n \geqslant 3$ und $E^G = \{e_1^G, \ldots, e_m^G\}$. Es wird angenommen, dass die Kanten so sortiert sind, dass $1 \leqslant i \leqslant d$ mit $n_1, \ldots, n_i = v_1$ und $n_{i+1}, \ldots, n_d \neq v_1$ sei. Falls nur Ersetzungsregeln in zusammenhängenden Graphen betrachtet werden, wird angenommen, dass die Knoten durch einen DFS-Aufruf nummeriert worden sind. Seien $\{v_1, \ldots, v_n\}$ genau die Knoten in dieser Reihenfolge. Dies stellt sicher, dass der Graph $G[\{v_2, \ldots, v_n\}]$ zusammenhängend ist. Im Folgenden wird definiert, wie eine Ersetzungsregel durch eine Ersetzungsregel in Normalform und durch eine Ersetzungsregel, in der der Ersetzungsgraph $n - 1$ Knoten aufweist, substituiert werden kann. Durch wiederholtes Anwenden solch einer Ersetzung kann sicherstellt werden, dass alle Ersetzungsregeln in Normalform vorliegen.

Die erste Ersetzungsregel erzeugte den Knoten v_1 und einen Knoten mit einem neuen Label ℓ_v. Der Knoten mit dem Label ℓ_v wird später durch den Restgraphen ersetzt. Die Anzahl an Kanten zwischen v_1 und dem Knoten mit dem Label ℓ_v entspricht der Anzahl an

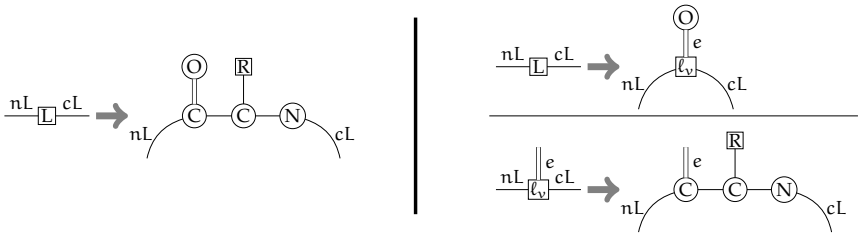


Abbildung 30: Normalisierung der Ersetzungsregeln

Die Ersetzungsregel auf der linken Seite der Abbildung wird durch zwei Ersetzungsregeln auf der rechten Seite ersetzt. Die erste Regel erzeugt den Knoten mit dem Label O und ein Knoten mit dem Label ℓ_v als Platzhalter für den verbleibenden Graphen. Die Kante zwischen diesen beiden Knoten weist das Label e auf, um sicherzustellen, dass sie am Ende den richtigen Endpunkt erhält. Die zweite Regel ersetzt den Knoten mit dem Label ℓ_v durch den verbleibenden Graphen und die Kantenlabel nL, cL sowie e und erhalten dadurch die korrekten Endpunkte.

Kanten des Graphen zwischen v_1 und dem Restgraphen, d. h. $|\delta(v_1)|$. Um sicherzustellen, dass alle Kanten die richtigen Endpunkte erhalten, werden Kantenlabel für alle diese Kanten eingeführt.

Sei ℓ_v ein Knoten- und seien $\ell_{e_1}, \ldots, \ell_{e_n}$ Kantenlabel. Die erste Ersetzungsregel einer Graph-Grammatik besteht aus dem Knoten v_1 mit seinem zugehörigen Label und einem neuen Knoten mit dem Label ℓ_v. Dieser Knoten ℓ_v wird später durch den Graphen $G[\{v_2, \ldots, v_n\}]$ ersetzt. Für jede Kante in $e \in \delta_E(v_1)$ wird eine Kante zwischen den beiden Knoten der Kante ℓ_e erzeugt. Sei der neue Endpunkt für die Kanten $e_1, \ldots, e_i$ der Knoten v_1 und für die Kanten $\{e_{i+1}, \ldots, e_d\}$ der Knoten mit dem Label ℓ_v.

Die zweite Ersetzungsregel einer Graph-Grammatik ersetzt das Knotenlabel ℓ_v, dessen inzidente Kanten die Label $(L_{i+1}, \ldots, L_d, (\ell_e)_{e \in \delta_G(v_1)})$ durch den Graphen $G[\{v_2, \ldots, v_n\}]$ aufweisen. Die neuen Endpunkte der Kanten mit den Labeln $L_{i+1}, \ldots, L_d$ sind die der ursprünglichen Ersetzungsregel. Endpunkte der Kanten mit den Labeln ℓ_e für $e \in \delta_E(v_1)$ unterscheiden sich von v_1 in G. Eine bildliche Beschreibung erhält man in Abbildung 30.

Die zweite Ersetzungsregel ist die einzige Ersetzungsregel, die das Knotenlabel ℓ_v ersetzen kann. Dies ist der Grund, weshalb nur beide Ersetzungsregeln gemeinsam angewendet werden können, um die ursprüngliche Regel zu substituieren.

Der Grad aller erzeugten Ersetzungsregeln kann durch die Summe des ursprünglichen Regelgrades sowie $\Delta(G)$ und $\zeta(G)$ in G be-

schränkt werden. Sei hierbei G der Ersetzungsgraph. Man unterscheidet zwischen dem Beitrag der Kanten, die den ursprünglichen Grad definieren und den zusätzlichen Kanten, bedingt durch die Ersetzung. Die ursprünglichen Kanten werden auf die zwei erzeugten Ersetzungsregeln aufgeteilt. Dies gewährleistet, dass die Kantenzahl in keiner Ersetzungsregel erhöht werden kann. Jede erzeugte Ersetzungsregel erzeugt einen Knoten von G oder einen Subgraphen von $G[\{v_i, \ldots, v_n\}]$. Im ersten Fall ist die Anzahl an zusätzlichen Kanten durch den Knotengrad beschränkt. Im zweiten Fall sind die Knoten $\{v_i, \ldots, v_n\}$ die Knoten im Teilbaum des DFS-Baumes, da die Knoten durch einen absteigenden DFS-Aufruf nummeriert worden sind. Hierbei stellt v_i die Wurzel des Baumes dar. Die Anzahl an Kanten aus G mit genau einem Endpunkt in $\{v_i, \ldots, v_n\}$, muss hierbei beschränkt werden. Man betrachte nun den DFS-Baum als ungerichteten Baum mit dem Knoten v_i als Wurzelknoten. Die Knoten eines jeden Teilbaumes von v_i, die nicht in $v_{i+1}, \ldots, v_n$ enthalten sind, bilden einen minimalen Schnitt, der maximal $\zeta(G)$ zum Grad der Ersetzungsregel beiträgt. Somit ist der Grad jeder Ersetzungsregel durch $\Delta(G) \cdot \zeta(G)$ beschränkt.

6

ALGORITHMUS ZUR MOLEKULAREN SUBSTRUKTURSUCHE

In diesem Kapitel wird ein Algorithmus vorgestellt, der in exponentieller Zeit alle möglichen Rückwärtssubstitutionen enumeriert, bis entweder das Startsymbol S abgeleitet worden oder keine Rückwärtssubstitution mehr möglich ist. Man speichert alle Subgraphen, die zum Erzeugen eines Labels genutzt werden in einer Tabelle, um Mehrfachaufzählungen der gleichen Struktur zu vermeiden. Der Algorithmus iteriert über alle Paare von molekularen Substruktur-Einträgen sowie alle bestehenden Graph-Grammatik-Regeln und versucht diese Regeln rückwärts anzuwenden. Kann durch Anwendung einer Graph-Grammatik-Regel auf den Graphen H der Graphen G hergeleitet werden, wird der Graph H als Vorgänger des Graphen G bezeichnet. Weisen Graph-Grammatik-Regeln und der Eingabegraph G einen beschränkten Grad auf, können alle möglichen Vorgänger H in polynomieller Laufzeit berechnet werden, indem man über alle Graph-Grammatik-Regeln iteriert und versucht, sie rückwärts auf alle Knotenpaare anzuwenden.

6.1 EXPONENTIELLER ALGORITHMUS

Seien $G = (V^G, E^G, N^G, L_V^G, L_E^G)$ der Anfragegraph und (S, P) ein Graphersetzungssystem. Seien $G(L_v, L_E(e_1), \ldots, L_E(e_d))$ der Sterngraph mit d Kanten und $L_E(e_1), \ldots, L_E(e_d)$ die Kantenlabel. Das Zentrum des Sterngraphen besitzt das Label L_v (die Label der anderen Knoten sind nicht spezifiziert).

Der Algorithmus speichert vom aktuellen Molekül (Graphen) folgende Werte in einer dynamischen Tabelle (DP). Sie ist wie folgt definiert:

$$DP(L_v, U, \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\} \in \{\mathtt{true}, \mathtt{false}\}.$$

Sei hierbei L_v ein Knotenlabel, U eine Teilmenge an Knoten, d der Grad einer Graph-Grammatik-Regel, die den Knoten L_V^G ersetzt und $\{L_E(e_1), \ldots, L_E(e_d)\}$ die Kantenlabel mit einem Endpunkt in U. Ein Tabelleneintrag in DP wird auf $\mathtt{true}$ gesetzt, falls es möglich ist, einen Subgraphen von $G[U]$, der die Kanten $\{e_1 \ldots, e_d\}$ aus $G(L_v, L_E(e_1), \ldots, L_E(e_d))$ enthält, zu erzielen.

Ein Subgraph vom Graphen G ist in $\mathcal{G}(S, P)$, falls ein Eintrag

$$DP(S, U, \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\}) = \mathtt{true}$$

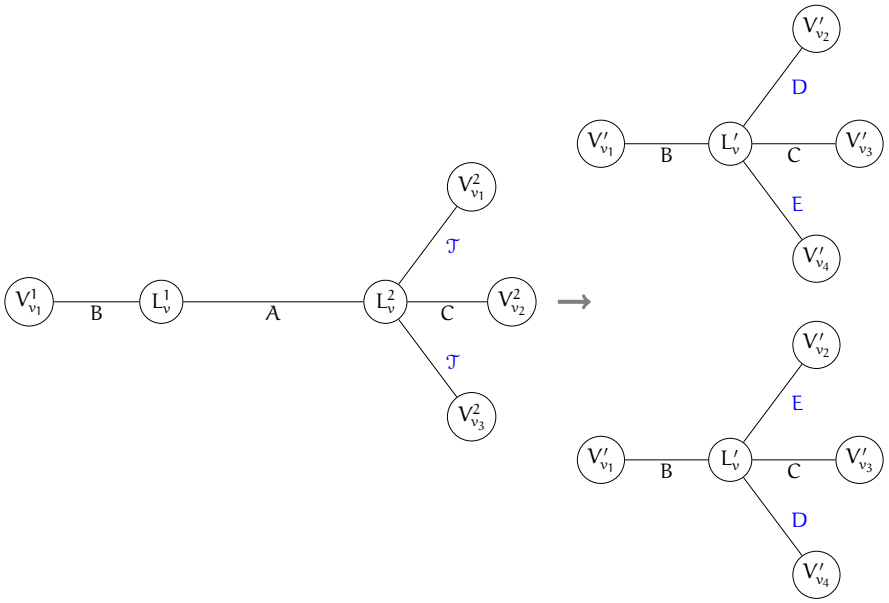


Abbildung 31: Kantenlabelkombinationen L_E

Die schwarzen Kantenlabel L_E sind bereits „fest“ mit Labeln belegt. Eine Regel p kann nur dann angewendet werden, falls für alle Kantenlabel $L_E(e_i), L_E(e_j)$ gilt, dass $i, j \in \{A, B, C, D, E\}$ und $i \neq j$. Die blauen Kantenlabel weisen im obigen Fall noch die initialen Label $\mathcal{T}$ auf, d. h. sie sind noch frei belegbar. Da alle true-Einträge in der DP bei der Erzeugung von weiteren true-Einträgen miteinander verglichen werden, müssen alle möglichen Kantenkombinationen abgespeichert werden, d. h. es werden alle Permutationen an noch zu belegenden Kantenlabeln ebenfalls in der DP abgespeichert. Im obigen rechten Fall bekommen die Kanten (L'_v, V'_{v_2}) und (L'_v, V'_{v_4}) jeweils einmal das Label D und einmal das Label E zugeteilt.

für das Startsymbol S, ein beliebiges U^H und beliebige Kanten $\{e^H_1, \ldots, e^H_d\}$ existiert.

Falls $\mathcal{G}(S, P)$ einen Subgraphen vom Graphen G enthält, müssen nur minimale Teilmengen U betrachten werden, d. h. falls

$$DP(L_v, U, \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\}) = \text{true},$$

kann auch jede Teilmenge $U' \supseteq U$ in

$$DP(L_v, U', \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\}) = \text{true},$$

gesetzt werden. Da diese Einträge auch nach einigen Iterationsschritten sowohl von $DP(L_v, U, \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\})$ als auch von $DP(L_v, U', \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\})$ auf true gesetzt werden können, werden diese Teilmengen nicht mit true markiert.

Am Anfang wird für jeden Knoten v eine Permutation an Kantenlabeln $L_E(e_1), \ldots, L_E(e_i)$ (ohne Wiederholung) jeweils ein Eintrag in der $DP(L_v, U, \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\})$ auf true gesetzt. Jedes Kantenlabel besitzt hierbei initial das Label $\mathcal{T}$. Alle anderen Werte werden auf false gesetzt. Sei $1 \leqslant i \leqslant |\delta(v)|$ und $\{e_1, \ldots, e_i\}$ die Kantenteilmengen der Kardinalität i, die inzident zu Knoten v sind. Es sei angemerkt, dass intuitiv ein Subgraph (V', E') durch Ableitung eines true DP-Eintrages der Knotenmenge V', der leeren Kantenmenge der DP-Einträge für die Knotenmenge $\{v\}$ und der Kantenmenge $\delta(v) \cap E'$ für alle $v \in V'$ ausgewählt wird. Sei eine Graph-Grammatik-Regel wie folgt definiert:

$$(L_v, L_{E_1}, \ldots, L_{E_d}) \to (H, n_1, \ldots, n_d).$$

Findet eine Regel Anwendung, werden die Kantenlabel $L_E(e_1), \ldots, L_E(e_i)$ entsprechend der Regel belegt. Kantenlabel, die einmal zugeordnet worden sind, können in den folgenden Iterationsschritten nicht mehr geändert werden. Weist ein DP-Eintrag mehrere Kantenlabel auf, die noch mit $\mathcal{T}$ gelabelt sind, erhalten alle freien Kantenlabel L_E eine Kantenlabelkombination aus den noch nicht fest vergebenen Kantenlabeln. Jede Kantenlabelkombination wird ebenfalls der DP hinzugefügt. Somit sind beim Vergleich der DP-Einträge alle nötigen Kombinationen an Kantenlabeln vorhanden (siehe Abbildung 31 für eine bildliche Beschreibung). Kantenlabel $\{e_1, \ldots, e_d\}$, die keinen Endpunkt in U aufweisen, finden bei einer Ableitung keine Beachtung.

Seien $V^H = U^1 \cup U^2$, $H = \{V^H, E^H\}$, $\bar{E} = \{e_1^1, \ldots e_{d_1}^1\} \cap \{e_1^2, \ldots, e_{d_2}^2\}$, $E^i = \{e_1^1, \ldots e_{d_1}^1\} \triangle \{e_1^2, \ldots, e_{d_2}^2\}$ und $U^1 \cap U^2 = \emptyset$. Ein weiterer true Eintrag wird erzeugt, indem über alle Paare von Einträgen

$$DP(L_v^1, U^1, \{e_1^1, \ldots, e_{d_1}^1\}, \{L_E^1(e_1^1), \ldots, L_E^1(e_{d_1}^1)\}),$$

$$DP(L_v^2, U^2, \{e_1^2, \ldots, e_{d_2}^2\}, \{L_E^2(e_1^2), \ldots, L_E^2(e_{d_2}^2)\}),$$

die bereits mit true markiert wurden, und alle Graph-Grammatik-Regeln P iteriert wird. Findet eine Regel Anwendung, wird sie auf den Graphen H angewendet, und ein neuer true Eintrag mit $DP(L_{v_i}^i, U^i = U^1 \cup U^2, E^i, L_E^i)$ wird zu DP hinzugefügt. Andernfalls wird das nächste Datenpaar aus DP verglichen. Durch Anwenden von Graph-Grammatik-Regeln werden alle möglichen Vorgänger H erzeugt. Eine Graph-Grammatik-Regel findet Anwendung, falls:

- die anzuwendende Graph-Grammatik-Regel P genau die Label $L_{v_1}^1$ und $L_{v_2}^2$ aufweist und zu $L_{v_i}^i$ ableitet,
- eine eineindeutige Zuordnung $m : \bar{E} \to E^i$ zwischen Kanten in $\bar{E}$ und Kanten im Graphen H, so dass $L_E(e) = L_E^i(m(e))$ für alle Kanten in $e \in \bar{E}$ besteht,

- eine eineindeutige Zuordnung $m' : E^1 \cup E^2 \to \{1, \ldots, d\}$ zwischen den Kanten $E^1 \cup E^2$ und den Indizes $1, \ldots, d$ besteht, so dass für eine Kante e von E^i, das Label $L_{m'(e)}$ und $n_{m'(e)} = u_i$ ist.

Der Algorithmus besitzt eine polynomielle Laufzeit, falls die DP eine Größe $|L^V| \cdot 2^{|V|} \cdot |E|^d$ aufweist und alle Regeln einen beschränkten Grad d aufweisen.

6.2 REDUZIERUNG DER ZU BERÜCKSICHTIGENDEN TEILMENGEN

Im Folgenden werden nur Regeln betrachtet, deren Graphen zusammenhängend sind. In diesem Fall induzieren die Mengen U immer nur verbundene Teilmengen des Graphen.

Sei $(L_v, U, \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\})$ ein true Eintrag in der dynamischen Tabelle DP. Werden die Kanten $\delta(U) \setminus \{e_1, \ldots, e_d\}$ aus G entfernt und der Graph besteht aus mehr als einer Zusammenhangskomponente, kann keine Regel anwendet werden, die Knoten aus unterschiedlichen Komponenten zusammenzieht. Daher können alle Knoten zu U hinzufügt werden, die aus der Komponente U nicht erreichbar sind (siehe Abbildung 32). In Abbildung 32 ist ein Graph G abgebildet, der die Knoten $v_1, \ldots, v_6$ und die Kanten $(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_4, v_5), (v_4, v_6)$ aufweist. Die Kanten- und Knotenlabel sind in diesem Beispiel nicht spezifiziert. Ein Datenbankeintrag DP weist hier für die Menge U (blaue Menge) die Knoten v_1, v_4 und die Kanten $e_1, e_2, (v_1, v_4)$ auf. Die Kanten aus $\delta(U)$ des DP-Eintrages, d. h. die Kanten e_1, e_2 stellen Verbindungen zu anderen Komponenten im Graphen G dar. In dieser Kantenkombination eines DP-Eintrages sind die Knoten v_3 und v_6 nicht erreichbar, da die rot gestrichelten Kanten in dieser Kombination nicht in dem aktuell betrachteten DP-Eintrag existieren, im Graphen G hingegen schon. Dadurch, dass kein weiterer Knoten im Graphen G mit dem Knoten v_6 verbunden ist, kann mit diesem DP-Eintrag keine Regel erzeugt werden, die den Knoten v_6 zusammenzieht. Aus diesem Grund wird der Knoten v_6 direkt der Menge U hinzugefügt und muss in den folgenden Iterationen vom aktuell erzeugten Eintrag nicht mehr beachtet werden. Der Knoten v_3 kann hingegen nicht direkt zur Menge U hinzugefügt werden, da er zusätzlich mit dem Knoten v_2 verbunden ist und somit in den folgenden Iterationen Regeln erzeugt werden können, die den Knoten v_3 verschmelzen. Im Folgenden werden nur Einträge

$$(L_v, U, \{e_1, \ldots, e_d\}, \{L_E(e_1), \ldots, L_E(e_d)\})$$

betrachtet, die die Menge U nicht auf diese Weise erweitern können.

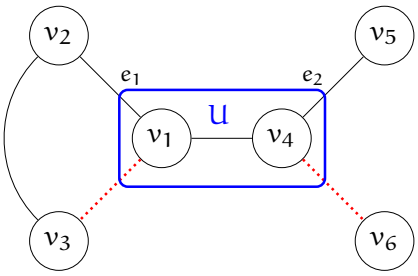


Abbildung 32: Resultierende Zusammenhangskomponenten

Falls der Eingabegraph G ein Baum ist, gibt es für jede Kantenteilmenge $\{e_1, \ldots, e_d\}$ höchstens eine Menge U an Knoten, den maximalen Teilbaum, der die Kanten $\{e_1, \ldots, e_d\}$ mit Blättern verbindet, falls solch ein Baum existiert (Ausnahme: Falls die Kantenmenge eine Kardinalität von 1 aufweist, existieren zwei mögliche Mengen). Daher weist der Algorithmus, falls der Grad der Regeln begrenzt ist, eine polynomielle Laufzeit auf Bäumen auf. In Kapitel 7 wird gezeigt, dass das Problem $\mathcal{NP}$-vollständig ist, falls die Regeln und der Anfragegraph einen unbeschränkten Grad aufweisen.

Eine Knotenmenge U kann durch die Kanten seines Schnittes $\delta(U)$ beschrieben werden (wiederum kann jede Kantenmenge zwei Knotenmengen bestimmen – genau die zwei Seiten des Schnittes). Da nur Teilmengen von U betrachtet werden, die nicht vergrößert werden können, kann die Teilmenge U eindeutig durch die Kanten $\{e_1, \ldots, e_d\}$ plus die Kanten auf dem Schnitt $\delta(U)$, die von $G[V \setminus U]$ von einem der Endpunkte von $\{e_1, \ldots, e_d\}$ erreichbar sind, die nicht in U sind, bestimmt werden. Hierbei muss für jede parallele Kante nur eine betrachtet werden.

Lemma 3.
Es müssen nur polynomiell viele Teilmengen U betrachtet werden, falls $\zeta(G) \cdot \Delta(G)$ durch Konstanten begrenzt sind. Es kann gezeigt werden, dass höchstens $\zeta(G) \cdot \Delta(G)$ Kanten benötigt werden, um den Schnitt zu beschreiben.

Beweis. Angenommen, es werden mehr als $\zeta(G) \cdot \Delta(G)$ Kanten benötigt, um die Menge U zu beschreiben. Dies bedeutet, dass mindestens eine Kante e_i existiert, für die es mehr als $\zeta(G)$ Kanten mit unterschiedlichen Endpunkten in $V \setminus U$ gibt, die durch einen Endpunkt u von e_i erreichbar sind, der im Graphen $G[V \setminus U]$ und nicht in U ist. Sei U' die Knotenmenge, die von u aus $G[V \setminus U]$ erreicht werden kann und U die andere Knotenmenge. Dies führt zum Widerspruch $\delta(U, U') > \zeta(G)$, da U und U' zusammenhängende Mengen darstellen. □

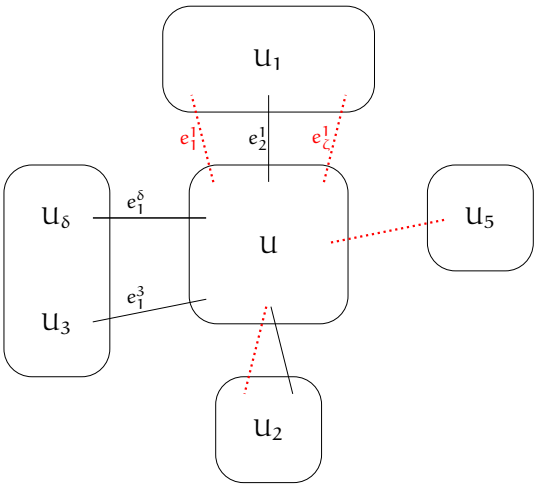


Abbildung 33: Maximale Anzahl an Kanten

In Abbildung 33 wird illustriert, dass maximal $\zeta(G) \cdot \Delta(G)$ Kanten benötigt werden, um $\delta(U)$ im Originalgraphen zu beschreiben. Der Graph weist die Komponenten $U, U_1, U_2, U_3, U_\delta, U_5$ auf. Die schwarzen Kanten sind im aktuellen DP-Eintrag enthalten, die rot gestrichelten hingegen nicht. Die rot gestrichelten Kanten bestehen nur im original Graphen. Seien $U_1, U_2, U_3, U_\delta, U_5$ die Komponenten in $G[V \setminus U]$. Die Komponenten U_1, U_2, U_3, U_δ sind im obigen Beispiel durch die schwarzen Kanten im aktuellen DP-Eintrag erreichbar, die Komponente U_5 hingegen nicht, da im aktuellen DP-Eintrag U keine Kante die Komponenten U und U_5 miteinander verbindet. Aus diesem Grund kann U_5, wie in Abbildung 32 definiert, zur Menge U hinzugefügt werden. Die Komponenten U_3 und U_δ sind jeweils über die Kanten e_1^δ und e_1^3 erreichbar. Wie in Kapitel 5 erläutert, kann $\Delta(G)$ durch den maximalen Grad einer Ersetzungsregel beschrieben werden, d. h. es existieren maximal $\Delta(G)$ viele Kanten in U. Der Schnitt U, U_1 weist im Originalgraphen genau drei Kanten e_1^1, e_2^1, e_ζ^1 auf, d. h. der Wert $\zeta(G)$ ist gleich drei. Da dieser Schnitt im obigen Beispiel die maximale Größe eines minimalen Schnittes darstellt, können die Schnitte zwischen U und U_i, d. h. zu allen anderen Komponenten, jeweils maximal die Größe des $\zeta(G)$-Wertes aufweisen. In Folge dessen kann $\delta(U \cup U_5)$ durch $\zeta(G) \cdot \Delta(G)$ Kanten im Originalgraphen nach oben abgeschätzt werden.

7 KOMPLEXITÄT DER SUBSTRUKTURSUCHE

In den nächsten Abschnitten wird die Komplexität des Problems zur Substruktursuche aufgezeigt. Das Subgraph-Matching-Problem ist $\mathcal{NP}$-vollständig, falls der Grad der Ersetzungsregeln und des Anfragegraphen nicht beschränkt wird. In Abschnitt 7.1 wird gezeigt, dass das Problem auch auf Bäumen, genauer, auf dem Sterngraphen $\mathcal{NP}$-vollständig bleibt. Dabei wird das 1in3SAT auf das Subgraph-Matching-Problem reduziert. Falls der Grad als Parameter genutzt wird, ist das Problem auch auf Bäumen $\mathcal{W}[1]$-schwer, dies wird in Abschnitt 7.2 aufgezeigt. In Abschnitt 7.3 wird bewiesen, dass auch auf Bäumen mit beschränktem Grad und beschränkter Baumweite das Problem $\mathcal{NP}$-vollständig bleibt.

7.1 $\mathcal{NP}$-VOLLSTÄNDIGKEIT AUF BÄUMEN

Das Graph- und Subgraph-Matching-Problem ist $\mathcal{NP}$-vollständig, falls der Grad der Ersetzungsregeln und des Anfragegraphen nicht beschränkt werden. Es wird gezeigt, dass diese Aussage auch für den Sterngraphen Gültigkeit findet. Ein Sterngraph ist ein Baum, der einen einzelnen internen Knoten, das Zentrum, aufweist. Das Zentrum ist durch Kanten mit allen anderen Knoten verbunden, während die anderen Knoten nur das Zentrum als Nachbarn besitzen.

Das 1in3SAT Problem wird auf das in Kapitel 6 gezeigte Problem reduziert. Sei $\varphi = c_1 \wedge \cdots \wedge c_m$ eine aussagenlogische Formel in konjunktiver Normalform, so dass jede Klausel c_i aus drei Literalen $c_i = l_1^i \vee l_2^i \vee l_3^i$ über einer Menge an Variablen $\{x_1, \ldots, x_n\}$ besteht, d. h. dass $l_j^i = x_k$ oder $l_j^i = \neg x_k$ ist. Man ist daran interessiert, ob es eine Zuweisung der Variablen gibt, so dass genau ein Literal für jede Klausel $\mathtt{true}$ ist. Schäfer [112] zeigte, dass dieses Problem $\mathcal{NP}$-vollständig ist. Auch Garey und Johnson [55] führen 1in3SAT als $\mathcal{NP}$-vollständiges Problem LO4.

Sei $\varphi = c_1 \wedge \cdots \wedge c_m$ mit $c_i = l_1^i \vee l_2^i \vee l_3^i$ eine aussagenlogische Formel über den Variablen $\{x_1, \ldots, x_n\}$. Sei $\{c_1, \ldots, c_m\} \cup \{l_j^i \mid 0 \leqslant i \leqslant n \text{ und } 0 \leqslant j \leqslant m\}$ die Menge der Knotenlabel und habe jede Kante das gleiche Label l_e. Sei der m-Sterngraph der Anfragegraph, dessen Zentrum das Label l_m^n aufweist. Die anderen Knoten besitzen die Label $c_1, \ldots, c_m$ (siehe Abbildung 34). Sei l_0^0 das Startlabel, das durch zwei Ersetzungsregeln substituiert werden kann. Eine entsprechende Belegung ist die Variable x_1 auf $\mathtt{true}$, die andere die Variable auf $\mathtt{false}$ zu setzen. Durch das Anwenden einer Regel, die der Belegung

$x_1 = \mathit{true}$ entspricht, werden die Nachbarn c_i für alle Klauseln mit dem Literal x_1 zum Zentrum hinzugefügt. Das Knotenlabel des Zentrums wird zu l_j^1 verändert. Hierbei entspricht j der Anzahl an Kanten, die inzident zum Zentrum sind, d. h. die Anzahl an Klauseln, die im ersten Schritt eine true Belegung erhalten haben. Solch ein Label kann durch die Regel ersetzt werden, deren x_1 Wert eine true Belegung aufweist. Der obere Index des Labels des Zentrums zeigt an, welche Variablen bereits einen true-Wert aufweisen. Der untere Index wird verwendet, um die Anzahl an Kanten zu zählen, die dem Sterngraphen hinzugefügt wurden, d. h. falls j-Klauseln ein Literal x_i besitzen, weist das Zentrum die neue Bezeichnung l_j^1 auf. Ein Label l_m^n des Sterngraphen besitzt genau m inzidente Kanten. Der erzeugte Graph matcht den Anfragegraphen, falls jedes Label c_i genau einmal erzeugt wurde und jede Klausel genau ein true Literal aufweist.

Die Regeln werden wie folgt definiert. Sei C_t^i der Index (t für true-Belegungen) aller Klauseln mit den Literalen x_i und C_f^i der Index (f für false-Belegungen) aller Klauseln mit den Literalen $\neg x_i$. Für jedes $0 \leqslant i < n$ und jedes $0 \leqslant j \leqslant m$ existieren die folgenden zwei Regeln:

- $(l_j^i, \underbrace{l_e, \ldots, l_e}_{j\text{-mal}}) \to (G, \underbrace{c, \ldots, c}_{j\text{-mal}})$, sei G der Graph mit dem Zentrum c mit Label $l_{j+|C_t^i|}^{i+1}$ und $|C_t^i|$ weiteren Knoten, einen mit Label c_k für jedes $k \in C_t^i$. Sei $r_j^{i,t}$ diese Regel.
- $(l_j^i, \underbrace{l_e, \ldots, l_e}_{j\text{-mal}}) \to (G, \underbrace{c, \ldots, c}_{j\text{-mal}})$, sei G der Graph mit dem Zentrum c mit Label $l_{j+|C_f^i|}^{i+1}$ und $|C_f^i|$ weiteren Knoten, einen mit Label c_k für jedes $k \in C_f^i$. Sei $r_j^{i,f}$ diese Regel.

Die Konstruktion kann eindeutig in polynomieller Laufzeit durchgeführt werden. Es ist zu zeigen, dass nur genau dann das 1in3SAT eine Lösung besitzt, falls ein Subgraph des Eingabesterngraphen durch das Graphersetzungssystem konstruiert werden kann.

Im Folgenden wird gezeigt, dass genau dann wenn das 1in3SAT eine Lösung aufweist, auch das aus Abschnitt 6.1 eine Lösung besitzt. Sei $\pi : \{x_1, \ldots, x_n\} \to \{\mathit{true}, \mathit{false}\}$ eine Belegung, so dass jede Klausel genau ein Literal enthält, das erfüllt ist. Beginnend mit Label l_0^0 und durch Anwenden der Regeln $r_0^{0,\pi(x_1)}, r_{|C_{\pi(x_1)}^i|}^{1,\pi(x_2)}, \ldots, r_{\sum_{i=1}^{n-1}|C_{\pi(x_i)}^i|}^{n-1,\pi(x_n)}$ erzeugt man am Ende den Eingabesterngraphen.

Es wird angenommen, dass ein Subgraph des Sterngraphen aus dem Startlabel l_0^0 unter Verwendung der Regeln konstruiert werden kann. Bei der Konstruktion erhält man immer einen Sterngraphen, dessen Zentrum das Label l_j^i aufweist und nur dieser Knoten und seine inzidenten Kanten ersetzt werden können. Es werden also genau

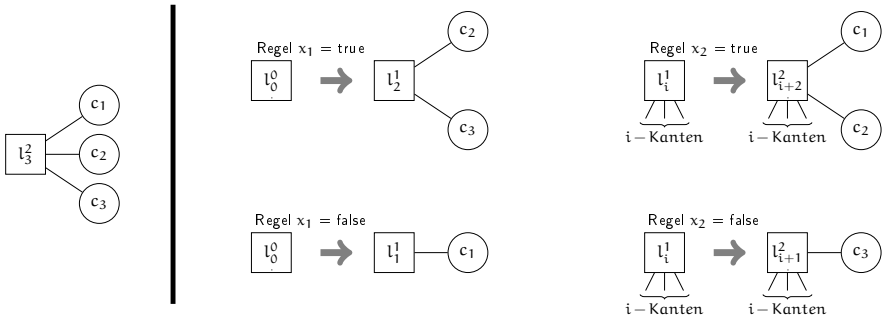


Abbildung 34: Die Konstruktion der 2-SAT Formel

Die Konstruktion aus der 2-SAT Formel $c_1 \wedge c_2 \wedge c_3$ mit $c_1 \equiv \neg x_1 \vee x_2$, $c_2 \equiv x_1 \vee x_2$ und $c_3 \equiv x_1 \vee \neg x_2$. Der Anfragegraph ist auf der linken und die Ersetzungsregeln auf der rechten Seite abgebildet. Mit der Regel, die x_1 auf true setzt, können die Label c_2 und c_3 zum Sterngraphen hinzugefügt werden. Ähnliches gilt für die anderen erfüllenden Belegungen. In diesem Beispiel ist es nicht möglich, Ersetzungsregeln, die eine erfüllende Belegung für x_1 und x_2 aufweisen, die den Sterngraphen mit den Labeln c_1, c_2, und c_3 für die Blätter und l_3^2 für das Zentrum zu erzeugen.

n Regeln verwendet, eine für jede Variable. Diese Regeln können als erfüllende Belegung interpretiert werden. Außerdem muss der untere Index des Labels des Zentrums immer genau der Kantenanzahl des Sterngraphen entsprechen. Da das letzte Label ein Label l_m^n aufweist, hat man am Ende den ganzen Eingabesterngraphen erzeugt (und nicht nur einen Subgraphen davon). Eine akzeptierende Belegung ist, falls jede Klausel genau ein Literal aufweist, das erfüllt ist.

Der untere Index des Labels l_j^i zählt die Anzahl an Knoten, um sicherzustellen, dass der komplette Sterngraph erzeugt wurde. Diese Konstruktion kann sowohl für das Subgraph- als auch für das Graph-Matching-Problem verwendet werden. Für das Graph-Matching-Problem kann der untere Index weggelassen werden.

7.2 $\mathcal{W}[1]$-VOLLSTÄNDIGKEIT

Der folgende Abschnitt zeigt auf, dass das Graph-Matching-Problem auch auf Bäumen $\mathcal{W}[1]$-schwer im Grade ist. Dies ist ein starker Hinweis darauf, dass es auch auf Bäumen schwierig sein wird, einen FPT-Algorithmus zu entwickeln. Eine Reduktion vom Longest-Common-Subsequence-Problem (dt. Längstes-Gemeinsames-Teilsequenz-Problem, LCSP) wird dies veranschauli-

chen. Pietrzak zeigte in [106], dass das LCSP $\mathcal{W}[1]$-schwer in der Anzahl an Sequenzen, auch für eine feste Größe an Alphabeten ihre Gültigkeit aufweist.

Im LCSP sind d Sequenzen $s_1, \ldots, s_d$ über einem Alphabet Σ und einer Zahl k gegeben. Die Frage ist: Gibt es eine Sequenz s der Länge k, die eine Teilsequenz von jedem der d Strings $s_1, \ldots, s_d$ ist. Es liegen folgende Label $\{C, c, t, h, e\} \cup \Sigma$ vor. Der Baum und die Regeln werden wie folgt erzeugt: Der Baum besteht aus einem zentralen Knoten, dem Zentrum, der das Label C aufweist. Ausgehend vom Knoten mit dem Label C entstehen d Pfade mit Labeln, die den Strings entsprechen. Das Abschlusssymbol der d genannten Strings stellt der Knoten mit dem Label e dar. Ebenfalls ausgehend vom Knoten mit dem Label C wird ein weiterer Pfad mit k Knoten erzeugt. Jeder dieser k Knoten weist das Label h auf. Das Abschlusssymbol dieses Strings ist der Knoten mit dem Label t. Eine Konstruktionsskizze ist in Abbildung 35 dargestellt.

Es existieren Regeln, die es ermöglichen, falls sie rückwärts betrachtet werden, einen Knoten, der ein Label aus Σ aufweist, in das Zentrum zu schrumpfen. Zusätzlich bestehen Regeln, die einen Sterngraphen mit d Knoten, die alle das gleiche Label aus Σ und einen Knoten mit Label h besitzen, ins Zentrum zu verschmelzen. Die Abschlussregel schrumpft das Zentrum mit d Knoten, die alle bis auf einen das Label e aufweisen (ein Knoten weist hierbei das Label t auf), zum Startsymbol zusammen (siehe Regeln rechts unten in Abbildung 35).

Im Folgenden wird argumentiert, wie ein Anfragegraph hergeleitet werden kann, falls eine Teilkette der Länge k existiert. Es wird gezeigt, in welcher Form der Anfragegraph durch Rückwärtssubstitutionen auf das Startlabel reduziert werden kann. Ausgehend vom Anfragegraphen werden alle Knoten ins Zentrum geschrumpft, die nicht Teil des gemeinsamen Teilstrings der Länge k sind. Es werden solange Knoten der gemeinsamen Teilzeichenfolge ins Zentrum geschrumpft, bis der erste verbleibende Knoten eines jeden Strings dem gemeinsamen Teilstring entspricht (in beliebiger Reihenfolge).

Im Anschluss werden d Knoten, die dem ersten Buchstaben des gemeinsamen Teilstrings entsprechen, und ein Knoten mit Label h ins Zentrum verschmolzen. Dieses Verfahren wird so lange fortgeführt, bis alle außer dem letzten Knoten ins Zentrum geschrumpft worden sind. Am Ende kann die Regel, die das Startlabel des Graphen erzeugt, angewendet werden.

Abschließend wird argumentiert, dass ein gemeinsamer Teilstring der Länge k existiert, falls der Subgraph erzeugt werden kann. In der nun folgenden Argumentation wird wieder die Vorwärtsrichtung der Regelanwendung verwendet. Im ersten Schritt kann nur die Regel

angewendet werden, die den Sterngraphen mit d Blättern mit dem Label e, ein Blatt mit dem Label t und das Zentrum mit dem Label c erzeugt. Durch diese Konstruktion ist es gewährleistet, dass neue Knoten nur durch diesen einzigen Knoten, der das Label c aufweist, erzeugt werden können. Daher kann nur der Graph und kein Subgraph gematcht werden, d. h. es müssen genau die vorgegebenen Strings erzeugt werden. Diese können entweder durch Hinzufügen eines einzelnen Buchstabens oder durch das Hinzufügen eines identischen Buchstabens zu allen Strings, außer zu einem Knoten, der die Bezeichnung h aufweist, erstellt werden. Man beachte, dass der Knoten mit der Bezeichnung h immer genau zum abschließenden Teil des Strings t hinzufügt werden muss, da andernfalls der Subgraph nicht erzeugt werden kann. Während der Erzeugung der Strings müssen k Knoten mit dem Label h erzeugt werden. Die entsprechenden Teilstrings erzeugen einen gemeinsamen Teilstring der Länge k.

Man beachte, dass die Regeln nur von d und dem Alphabet Σ abhängen, d. h. dass das Härteergebnis auch für eine unveränderliche Graph-Grammatik gilt. Des Weiteren werden keine Kantenlabel in der Normalform der Regeln benötigt, indem jedem String eine eindeutige Kopie des Alphabets übergeben wird.

Die Konstruktion kann dazu verwendet werden, die Härte des Subgraph-Matching-Problems aufzuzeigen.

Man beachte, dass das Graph-Matching-Problem in der Klasse Slice-wise polynomial für alle Graphen mit beschränktem Grad enthalten ist, während Subgraph-Matching-Algorithmen aus dem Kapitel 6 dafür durch einen zusätzlichen Parameter beschränkt werden müssen.

7.3 $\mathcal{NP}$-VOLLSTÄNDIGKEIT AUF GRAPHEN MIT BESCHRÄNKTEM KNOTENGRAD UND BESCHRÄNKTER BAUMWEITE

Dieser Abschnitt erläutert, warum das Subgraph-Matching-Problem auch auf Graphen mit beschränktem Grad und beschränkter Baumweite $\mathcal{NP}$-vollständig bleibt. Das Ergebnis dieses Abschnittes schließt die Baumweite als Parameter für das in Kapitel 6 gezeigte Problem aus.

Um zu zeigen, dass der Graph-Grammatik-Ansatz auch auf Graphen mit beschränktem Knotengrad und beschränkter Baumweite $\mathcal{NP}$-vollständig bleibt, wird eine Reduktion vom bounded Post's correspondence problem (dt. beschränktes Post'sches Korrespondenzproblem, BPKP) auf den Graph-Grammatik-Ansatz durchgeführt, das wie folgt definiert ist: Seien n Paare an Strings $(x_1, y_1), \ldots, (x_n, y_n)$ gegeben und sei $K \leqslant n$. Gefragt ist: Existiert eine Sequenz $i_1, \ldots, i_k$

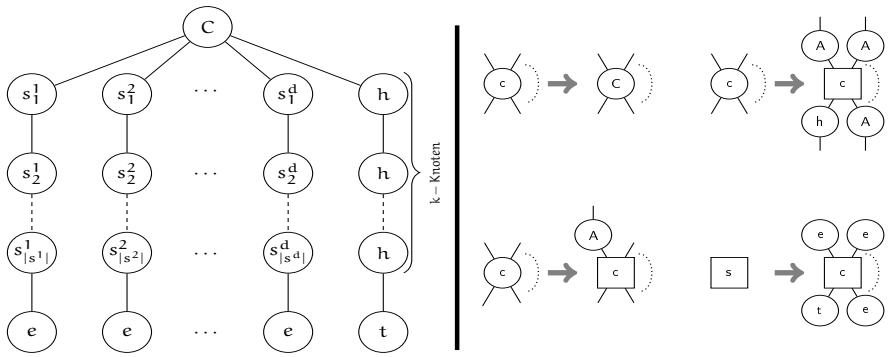


Abbildung 35: Graph- und Regelerzeugung – $\mathcal{W}[1]$-Vollständigkeit
Die Skizze des Eingabegraphen auf der linken und die Regeln auf der rechten Seite. Die Regeln müssen für alle $A \in \Sigma$ erstellt werden. Die Regel oben links ersetzt einen beliebigen String durch ein beliebiges Symbol. Die Regel unten links erweitert einen beliebigen String um ein beliebiges Symbol. Die Regel oben rechts erweitert alle Strings um das gleiche Symbol und der Zähler für die Anzahl an übereinstimmenden Symbolen wird um eins erhöht (dies ist der Pfad der Länge k, der h Knoten aufweist). Die Regel unten rechts erzeugt das Startlabel.

mit $k \leqslant K$ an Indizes (nicht notwendigerweise verschieden), so dass $x_{i_1} x_{i_2} \dots x_{i_k} = y_{i_1} y_{i_2} \dots y_{i_k}$ gilt. Die $\mathcal{NP}$-Vollständigkeit des BPKP wurde durch Hunt et al. [71] und in [55] gezeigt. Es sei darauf hingewiesen, dass die nachfolgende Reduktion für $k = K$ gilt. Es ist offensichtlich, dass falls das Problem für $k \leqslant K$ $\mathcal{NP}$-vollständig ist, dies auch für $k = K$ gilt.

Um den komplizierten Aufbau des Reduktionsgraphen besser darzustellen, wird er anhand eines Beispiels präsentiert. Das BPKP in diesem Beispiel weist die folgenden Stringpaare auf: (D, DE), (ED, D). Die Lösung des Problems ist offensichtlich, d. h. werden die Sequenzen $(1, 2)$ miteinander verbunden, ergeben sie die folgenden $DED = DED$ Wörter für die x- und y-Strings.

Sei eine Instanz des BPKP gegeben. Daraus wird folgender Graph und Grammatik konstruiert: Der Graph besteht zentral aus dem Knoten C. Links von ihm folgt ein b-Knoten und K Kopien der Strings $x_1 x_2 \dots x_n$. Jeder String x_i besteht hierbei aus $|x_i|$ Elementen, wobei jedes Element durch einen eigenen Knoten dargestellt wird. Die Strings werden jeweils am Ende durch einen zusätzlichen Knoten b separiert, um den Anfang und das Ende eines jeden Strings zu beschreiben. Am Ende einer Sequenz folgt ein Knoten mit dem Label e_x. Der Eingabeparameter besteht in diesem Fall aus $K = 2$ Strings, aus diesem

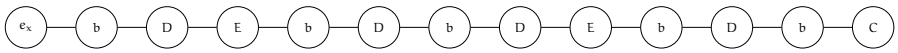


Abbildung 36: Erstellung des Teilgraphen der Sequenz für die x-Strings

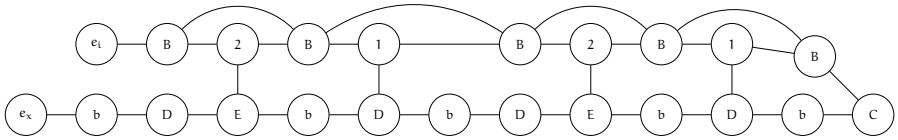


Abbildung 37: Indexknoten eines Strings

Grund werden die vorhandenen Strings zweimal hintereinander, wie beschrieben notiert (siehe Abbildung 36).

Der erste Knoten eines Strings x_i ist zusätzlich mit genau einem Knoten mit dem Label i, der den Index des jeweiligen Strings angibt, verbunden. Wieder beginnend beim zentralen Knoten C werden zwischen diesen Indexknoten Knoten mit der Bezeichnung B hinzugefügt. Diese trennen die Indexknoten im oberen Kontrollstring voneinander. Zusätzlich werden diese B-Knoten mit einer Kante verbunden, um später sicherzustellen, dass ein Pfad zwischen den ausgewählten Strings der Lösungsmenge besteht (siehe Abbildung 37).

Wie für die x-Sequenz werden auf der rechten Seite des Graphen die Sequenz und dessen Kontrollstrukturen auch für die y-Werte, wie beschrieben, aufgebaut. Die rechte Seite des Graphen besteht ebenfalls aus K Kopien der Strings $y_1 y_2 \dots y_n$. Jedes y_i ist auch hier mit einem zusätzlichen Knoten mit dem Label i verbunden.

Des Weiteren werden weitere Kontrollstrukturen eingeführt, die sicherstellen, dass die Anzahl an verbleibenden Strings von x und y übereinstimmen und zusätzlich werden mit ihnen die Anzahl an Strings von x und y gezählt. Jeder dieser Knoten wird mit c_x beziehungsweise c_y bezeichnet. Es werden genau $K + 1$ dieser Knoten benötigt. Diese Kontrollstrukturen werden abschießend mit dem Knoten e notiert (siehe Abbildung 38).

Zusätzlich wird im Graphen ein weiterer Kontrollstring eingefügt. Dieser stellt sicher, dass die gewählten Indexsequenzen in der Lösungssequenz identisch und genau K lang sind. Jeder dieser Knoten wird mit dem Label c_i markiert, enthält genau K Knoten und schließt mit dem Knoten e ab (siehe finaler Graph in Abbildung 39).

Die Transformation des Reduktionsgraphen erfolgt offensichtlich in polynomieller Zeit $\mathcal{O}(n \cdot K \cdot L)$, wobei L die Länge des längsten Teil-

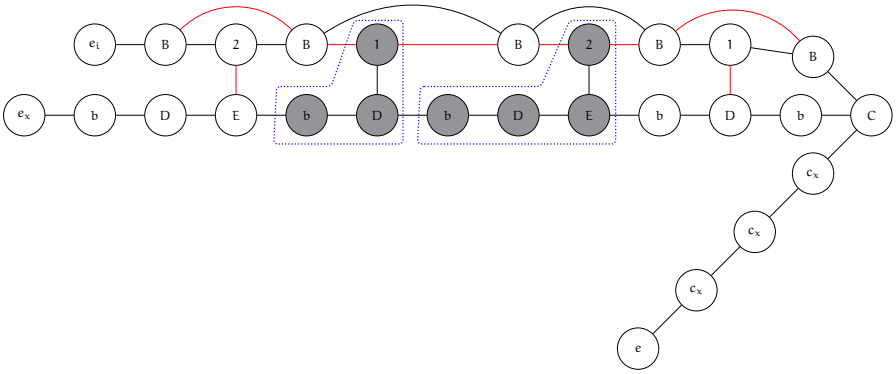


Abbildung 38: Kontrollknoten zur Überprüfung der verbleibenden Strings

strings darstellt. Es muss des Weiteren gezeigt werden, dass genau dann eine Lösung für den Graph-Grammatik-Ansatz existiert, wenn eine für das BPKP vorliegt.

Es existiert eine Regelmenge, die Folgendes erlaubt: Eine Regel löscht entweder einen String x_i und den angehängten Indexknoten i oder gar keinen Knoten dieses Teilgraphen (siehe blau umrandete Teilgraphen in Abbildung 38). Weitere Regeln sorgen dafür, dass die entsprechenden Knoten der Strings und die entsprechenden Knoten der Label übereinstimmen. Die Konstruktion des Graphen wird durch die Tatsache verkompliziert, dass die Positionen der Indizes in x und y unterschiedlich sind. Es existieren Regeln, die erlauben, einen Index zu überspringen, dessen Zeichenfolge noch in der unteren Sequenz ist. Das muss ausgeschlossen werden. Daher wird die Anzahl der entsprechenden Indizes und die Anzahl der Zeichenfolgen in x und y gezählt, was sicherstellt, dass sie übereinstimmen. Daher wurden vier unterschiedliche Regelklassen eingeführt:

- Regeln zum Löschen einer Zeichenfolge x_i und y_i, deren Knoten den Index angeben,
- Regeln, die sicherstellen, dass die verbliebenen Strings von x und y übereinstimmen und zusätzlich die Anzahl der Strings von x und y zählen,
- Regeln, die die Länge der übereinstimmenden Subsequenzen der verbleibenden Indizes in x und y zählen,
- Regeln, die sicherstellen, dass diese Subsequenz die gleiche Länge aufweist, wie die Anzahl an verbleibenden x- und y-Strings und dass ihre Länge mindestens eins ist.

Jede Lösung $(1,\ 2)$ stellt hier eine Teilsequenz der „Indexsequenz“ dar. Genaueres Vorgehen anhand des obigen Beispiels: Um das

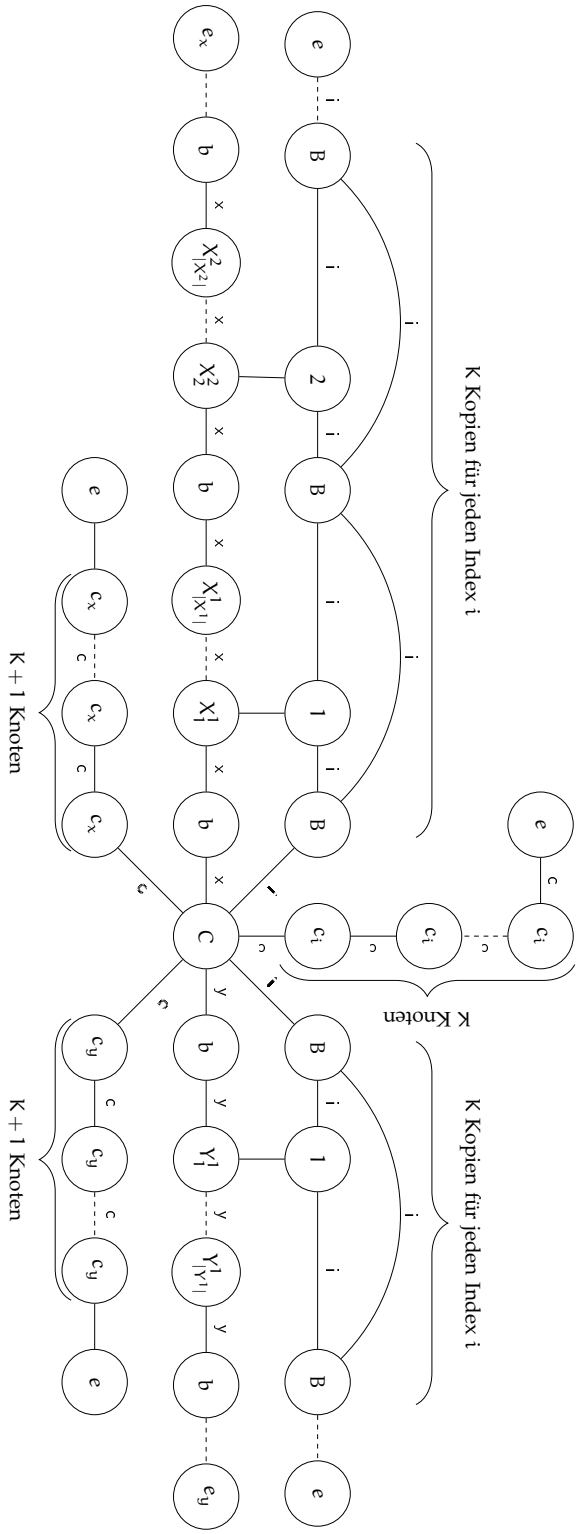


Abbildung 39: Kompletter Reduktionsgraph

Subgraph-Matching zu lösen, werden nun Knoten nach einem festen Regelsatz verschmolzen und Kanten entfernt. Am Ende soll hierbei genau der Subgraph gematcht werden, der die Knoten und Kanten aufweist, die bestehen bleiben. Wird nun ein Index ausgewählt, der im Lösungstupel ist, wird die Kante, die zwischen dem Indexknoten i und dem ersten Symbol des Strings ist, gelöscht. Wird ein Index nicht gewählt, werden die Kanten zwischen den nicht gewählten Indexknoten und den B-Knoten entfernt. In diesem Beispiel werden die rot markierten Kanten gestrichen (siehe Abbildung 38). Für den y-String werden die entsprechenden Kanten entfernt. Man muss darauf achten, dass die i-Strings in der Reihenfolge des Lösungstupels sukzessive ausgewählt werden, d. h. am Anfang muss ein String mit dem Index eins ausgewählt werden, anschließend darf nur ein String mit dem Index zwei ausgewählt werden. Ob dies direkt nach dem ersten gewählten String erfolgt oder erst später, ist unerheblich. Die Teilgraphen (Teilstrings inklusive den kleinen bs), die nicht ausgewählt wurden, werden mittels Graph-Grammatik-Regeln zusammengeschrumpft (siehe grau markierte Knoten in Abbildung 38).

Die Erstellung der Graph-Grammatik-Regeln zum Zusammenschrumpfen der nicht gewählten Indizes, inklusive deren Knoten und Kanten, wird nachfolgend beschrieben:

- Regel 1: Verschmilzt einen Indexknoten mit dem ersten Symbolknoten der jeweiligen Sequenz zu einem Knoten mit dem Label L.
- Regel 2: Verschmilzt alle noch verfügbaren Symbolknoten einer Sequenz, bis keiner mehr vorhanden ist.
- Regel 3: Verschmilzt den Knoten mit dem Label L mit den zwei verbliebenen Kontrollknoten einer Sequenz zu einem Kontrollknoten, der mit dem restlichen String des jeweiligen x- oder y-Strings verbunden ist.

Für eine bebilderte Beschreibung der aufgezählten Regeln sei auf Abbildung 40 verwiesen. Die oberste stellt Regel 1, die mittlere Regel 2 und die unterste Regel 3 dar.

Man startet in Knoten C. Auf der rechten und linken Seite im Sequenzstring folgen als erstes jeweils bs. Diese können nur mit dem Knoten C verschmolzen werden, falls noch Kontrollknoten c_x aus x oder c_y aus y vorhanden sind, d. h. dass sowohl die Anzahl an c_x- als auch c_y-Knoten ist größer als eins. Da aktuell noch ausreichend $c_x = 3$ und $c_y = 3$ Knoten vorhanden sind (weil noch keine dieser Knoten verschmolzen wurden), können die ersten bs auf beiden Seiten mit dem Knoten C verschmolzen werden und es wird anschließend ein Knoten der jeweiligen Kontrollstruktur c_x und c_y um eins

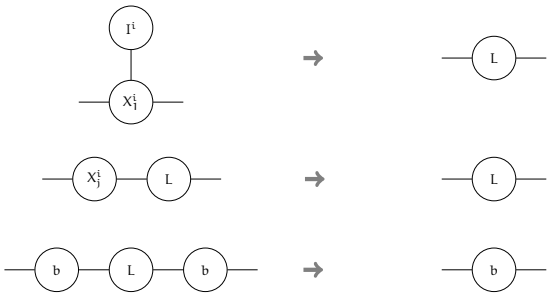


Abbildung 40: Graph-Grammatik-Regeln zum Zusammenschrumpfen der nicht benötigten Sequenzen

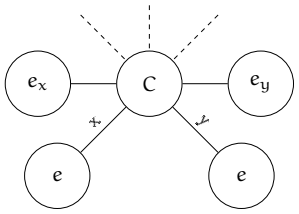


Abbildung 41: Reduzierter BPKP-Graph

vermindert, indem jeweils ein Knoten c_x und c_y mit dem Knoten C zu C verschmolzen wird. Folgen auf beiden Seiten der Sequenzen nun die gleichen Symbole, können sie ohne weiteres auch mit dem Knoten C verschmolzen werden, wie z. B. der erste Knoten D im x- und y-String. Es wird so lange wie beschrieben über die Strings von x und y iteriert, bis man entweder das Ende des jeweiligen Strings erreicht hat oder keine Möglichkeit mehr besteht, entweder gleiche Symbole in beiden Strings an der gleichen Position zu verschmelzen oder es nicht mehr ausreichend Kontrollknoten von x oder y zur Verfügung stehen, d. h. sowohl die Anzahl der verbliebenen c_x- als auch c_y-Knoten gleich null ist, um weitere bs zusammenführen zu können. Am Ende einer gültigen Lösung sollten nur noch die Knoten, wie in Abbildung 41 im Graphen, Bestand haben.

Es muss nun überprüft werden, ob die gleichen Teilstrings und die gleiche Anzahl dieser ausgewählt wurden. Dies wird wie folgt sichergestellt: man folgt, wieder ausgehend vom Knoten C, den oberen Pfaden. Hierbei wird alternierend in einer gültigen Lösung ein Knoten B und Indexknoten besucht. Die Bs können ohne weiteres in den Knoten C hineingeschrumpft werden. Folgen auf beiden Seiten von x und y die gleichen Indexknoten, werden diese ebenfalls mit dem Knoten C verschmolzen. Im gleichen Schritt werden die Kontrollkno-

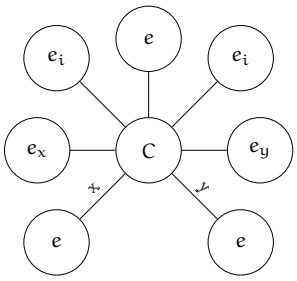


Abbildung 42: Finaler Graph einer gültigen Lösung

ten c_i um eins vermindert, indem ein Knoten c_i mit dem Knoten C zu C verschmolzen wird. In einer final gültigen Lösung sollten im Graphen die Knoten wie in Abbildung 42 verbleiben, d. h. alle Knoten der Kontrollstrukturen, bis auf deren Endknoten, müssen verschmolzen worden sein. Dieser verbleibende Graph kann anschließend zum Startsymbol abgeleitet werden. Nur wenn eine gültige Lösung gefunden wurde, also das Startsymbol abgeleitet wurde, ist eine Lösung für das BPKP gefunden worden und vice versa.

7.4 SCHÄTZUNG DER DP-TABELLENGRÖSSE IN EINER DATENBANK

In Abschnitt 6.2 wurde gezeigt, dass eine Graphenklasse, die durch den Parameter $\zeta(G)$ beschränkt worden ist, eine polynomielle Laufzeit Laufzeit aufweist. Die Größe der DP-Tabelle für einen Graphen $G = (V, E)$ kann mit $|\mathcal{L}^V| \cdot |E|^{\Delta(G)\cdot\zeta(G)}$ abgeschätzt werden. Dieser Wert wurde für alle All-Purchasable-Structures[1] (dt. alle käuflichen Strukturen) in der Zinc-Datenbank berechnet und ausgewertet. Die Analyse wird in diesem Abschnitt in einer Statistik zusammengefasst. Die Datenbank enthält etwa 22 Millionen Strukturen und wird oft in der computergestützten Wirkstoffforschung genutzt. Der Modus des Parameters $\zeta(G)$ über die Strukturen beträgt zwei. Für die überwiegende Mehrheit der Strukturen beträgt der Parameter maximal sechs und weist einen maximalen Wert von 22 auf. Tabelle 4 zeigt die Verteilung des Wertes $\zeta(G)$.

Sei s die Anzahl an Teilmengen U an Knoten, so dass $G[U]$ und $G[V \setminus U]$ verbunden sind. Die Größe der DP-Tabelle kann als Folge dessen durch $|\mathcal{L}^V|s^{\Delta(G)}$ beschränkt werden, d. h. falls die Eingabegröße polynomiell und der maximale Grad beschränkt ist, folgt eine polynomielle Laufzeit. Für die meisten Instanzen liegt der Wert s zwischen 16 und 2.048. Nur 2‰ weisen einen höheren Wert auf. 209

1 http://zinc.docking.org/subsets/all-purchasable

Tabelle 4: Verteilung des $\zeta(G)$-Wertes
Die Tabelle stellt die Verteilung des $\zeta(G)$-Wertes dar. Dabei ist zu erkennen, dass die Elemente der Werte $1-6$ ca. 99% der gesamten Verteilung ausmachen. Hierbei wurden ca. 22 Millionen Strukturen untersucht.

Anzahl	1	2	3	4	5	6	7	8	9	10	11
ζ	105.329	13.429.106	4.027.109	3.450.440	541.304	308.973	62.943	25.141	5.649	1.698	518
Anzahl	12	13	14	15	16	17	18	19	20	21	22
ζ	268	152	72	29	35	11	10	1	2	0	1

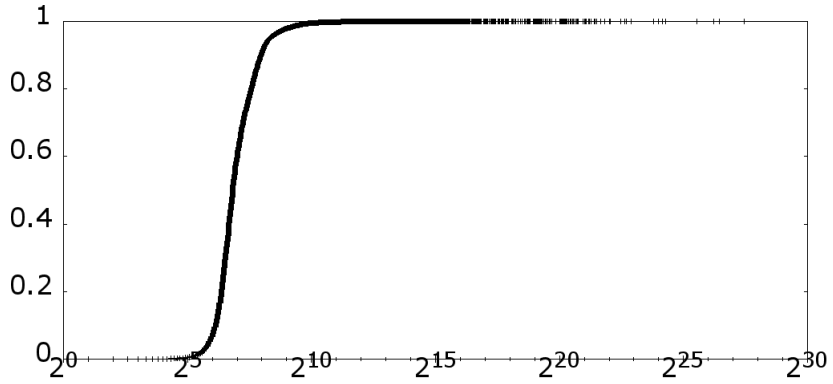


Abbildung 43: Verteilung der Strukturen in G
Das Diagramm zeigt auf der y-Achse die prozentuale Verteilung der Strukturen in G. Der x-Achse entnimmt man die Anzahl an Teilmengen U an Knoten, mit denen $G[U]$ und $G[V \setminus U]$ verbunden sind. Für mehr als 2‰ der Strukturen liegt der Wert zwischen 16 und 2048.

Instanzen besitzen einen Wert von über 100.000 und nur zwei Instanzen weisen mehr als 2 Millionen Komponenten auf. Eine Auswertung der Daten erhält man in Abbildung 43. Man beachte, dass es sich bei der obigen Abbildung um eine Worst-Case-Abschätzung handelt. Die DP-Tabelle weist in der Praxis einen wesentlich geringeren Umfang auf.

Strukturen wie Fullerene [129] weisen einen hohen ζ-Wert auf. Daraus folgt, dass der Algorithmus aus Kapitel 6 bei diesen Strukturen nicht effizient ist. Diese Strukturen sind typischerweise auch für andere Abfragesprachen wie SMARTS aufwendig zu berechnen. SMARTS nutzen Vorberechnungen, wie z. B. die auf die Eigenschaft des Smallest-Set-of-Smallest-Rings (dt. kleinste Menge an kleinsten Ringen) [128] beruhen.

7.5 ZUSAMMENFASSUNG DER THEORETISCHEN ERGEBNISSE ZUR SUBSTRUKTURSUCHE

In den Abschnitten 7.1 und 7.3 wurde gezeigt, dass das hier vorgestellte Problem $\mathcal{NP}$-vollständig bleibt, auch falls der Eingabegraph einen Sterngraphen darstellt. Es wurde gezeigt, dass das Subgraph-Matching-Problem für einen Graphen G und ein Graphersetzungssystem (S, P) für den Fall, dass $\Delta(G)$, $\zeta(G)$, $\Delta(S, P)$ und $\zeta(S, P)$ beschränkt sind, in polynomieller Zeit gelöst werden kann. Für jeden Baum T ist $\zeta(T) = 1$. Der vorgestellte Algorithmus ist kein FPT-Algorithmus im Grad und der maximalen Größe des minimalen Schnittes. In Abschnitt 7.2 wurde gezeigt, dass das untersuchte Problem auch auf Bäumen $\mathcal{W}[1]$-schwer ist und daher kein FPT existiert kann sofern $\mathcal{W}[1] = \mathrm{FPT}$ gilt. In Abschnitt 7.3 wurde bewiesen, dass das Problem auch auf Graphen mit beschränktem Grad und beschränkter Baumweite $\mathcal{NP}$-vollständig bleibt. Daher kann dieser Parameter zur Beschreibung der Komplexität eines Graphen nicht für das vorgestellte Problem verwendet werden.

In Abbildung 44 sind die erzielten Ergebnisse zum Subgraph-Matching-Problem zusammengefasst worden. Der vorgestellte Algorithmus besitzt polynomielle Laufzeit, falls die Größen der Parameter begrenzt sind (XP-Algorithmus). Somit kann der Algorithmus auch auf spezielle Graphenklassen angewendet werden, während die Härteergebnisse ($\mathcal{NP}$-Vollständigkeit ($\mathcal{NPC}$) und $\mathcal{W}[1]$-Härte) auch auf größere Graphenklassen generalisiert werden können. Die Hochzahlen zeigen die Abschnitte an, in welchem Abschnitt die jeweiligen Ergebnisse vorgestellt worden sind. Alle Härteergebnisse, außer die aus Abschnitt 7.3 gelten auch für eine beschränkte Menge an Ersetzungsegeln. Der Algorithmus in Kapitel 6 kann auch verwendet werden, falls die Ersetzungsregeln Teil der Eingabe sind. Für das Graph-Matching-Problem ist der Algorithmus von Lautemann [85] ein XP-Algorithmus für alle Graphen mit beschränktem Grad. Der entsprechende $\mathcal{NP}$-Beweis wird nicht in dieser Arbeit übernommen. Clemens Lautemann [85] entwickelte 1990 einen Algorithmus für das Graph-Matching-Problem unter der Voraussetzung, dass der Grad des Eingabegraphen beschränkt ist. In seinem Algorithmus war es nicht nötig einen weiteren Parameter zu fixieren. Es wurde gezeigt, dass das Subgraph-Matching-Problem in diesem Fall $\mathcal{NP}$-schwer bleibt und es daher unwahrscheinlich ist, dass der Algorithmus generalisiert werden kann, ohne damit gleichzeitig die Laufzeit zu erhöhen. Dennoch kann der hier vorgestellte Algorithmus als seine Verallgemeinerung des von Lautemann entwickelten Algorithmus angesehen werden.

Die Verbindungsersetzung aus dem NRG-Verfahren aus Abschnitt 2.3.1 ist kongruent zu dem vorgestellten aus Kapitel 6. Der Unterschied zwischen dem NRG und dem Verfahren ist, dass das Ver-

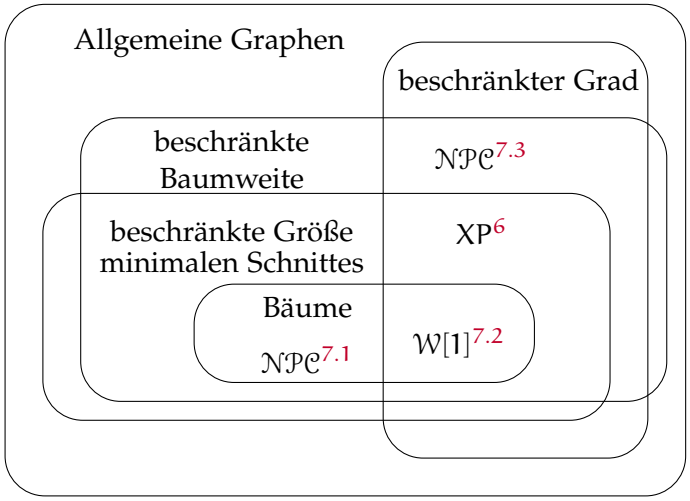


Abbildung 44: Überblick der erzielten Ergebnisse zum Subgraph-Matching-Problem.

fahren den Kanten explizit neue Endpunkte zuweist. Die Ersetzungsregeln aus Kapitel 6 können als HRG-Ersetzungsregeln im dualen Hypergraphen, mit der zusätzlichen Bedingung, dass jeder Knotengrad genau zwei beträgt (d. h. der duale Graph eines Hypergraphen ist ein Graph), betrachtet werden.

Der Hauptteil des ersten Teils dieser Arbeit befasste sich mit der Entwicklung eines Algorithmus zur Substruktursuche unter Zuhilfenahme von Graph-Grammatiken und untersuchte die Komplexität des vorgestellten Ansatzes. Der folgende, zweite Teil dieser Arbeit, stellt Algorithmen vor, die aus einer bestehenden Menge an Molekülen Graph-Grammatiken erlernen. Es werden mehrere Ansätze aufgezeigt.

8

ALGORITHMEN ZUM LERNEN VON GRAPH-GRAMMATIKEN

Dieses Kapitel behandelt die Konstruktion, Verbesserung und Evaluierung von Algorithmen zum Lernen von Graph-Grammatiken anhand von Beispielen. Es werden unterschiedliche Ansätze vorgestellt. Ziel dieses Kapitels ist es, Graph-Grammatiken zu generieren, die zum einen eine gewünschte Substruktur finden und zum anderen die Möglichkeit bieten, Moleküle zu klassifizieren. Zum Lernen der Graph-Grammatik-Regeln wird eine Lernmenge L und eine Testmenge T an Molekülen benötigt, wobei $L \cap T = \emptyset$ gilt. Die Daten der Lern- und Testmenge werden im SMILES-Format (siehe Abschnitt 2.1.1) als Eingabe an die Algorithmen aus den Abschnitten 8.1, 8.2 und 8.3 übergeben. Diese wurden zufällig aus der Zinc-[1] und PubChem[2] Datenbank entnommen. Die Moleküle der Lernmenge gehören entweder der gewünschten funktionalen Gruppe an oder sie weisen alle eine gemeinsame Substruktur auf.

Der erste Ansatz in Abschnitt 8.1 ist ein naiver Algorithmus. Dieser erzeugt aus einer Menge an gegebenen Molekülen eine Menge an unterschiedlichen Graph-Grammatik-Regeln. In Abschnitt 8.2 wird ein verbesserter Ansatz vorgestellt. In Abschnitt 8.3 wird eine Methode präsentiert, welche chemische Aspekte bei der Erzeugung der Regeln einbezieht. In Abschnitt 8.4 wird eine gegebene Regelmenge generalisiert. Es wird gezeigt, dass durch die Generalisierung weniger Graph-Grammatik-Regeln benötigt werden, um Moleküle der gesuchten Klasse hinzuzufügen oder eine gewünschte Substruktur aufzufinden. Bei allen Ansätzen dieses Kapitels werden die Kantenlabel bei der Regelerzeugung aufgrund der Generalisierung auf eine Menge an Molekülen nicht betrachtet. Die Nutzung von Kantenlabeln würde die gewünschte Zuordnung einschränken und die Anzahl an Regeln erhöhen. Zusätzlich werden in Abschnitt 8.4 die Knotengrade vernachlässigt, somit können Graph-Grammatik-Regeln weiter generalisiert werden. Am Ende werden die unterschiedlichen Ansätze miteinander verglichen und evaluiert.

8.1 NAIVER ALGORITHMUS ZUM LERNEN VON GRAPH-GRAMMATIKEN

Der naive Algorithmus versucht für jede Kante im Graphen G eine normalisierte Regel zu erstellen, d. h. für jedes Molekül G in der Lern-

1 **http://zinc.docking.org/subsets/**.
2 **https://pubchem.ncbi.nlm.nih.gov/**.

menge L berechnet der Algorithmus die dazugehörigen normalisierten Graph-Grammatik-Regeln (siehe Kapitel 5) rekursiv, wobei die Kantenlabel bei der Erzeugung der Regeln aus den bereits genannten Gründen der Generalisierung keine Beachtung finden. Die erzeugten Regeln werden alle gemeinsam in einem Set gespeichert und anschließend zur Klassifikation oder Substruktursuche eingesetzt.

Sei $G = (V_G, E_G)$ ein ungerichteter, zusammenhängender Graph, V_G die Knoten- und E_G die Kantenmenge. Sei $v \in V_G$, $e \in E_G$ und $N(v)$ die Menge aller Nachbarn des Knotens v, $H = (V_H, E_H)$ ein Teilgraph von G, $|V|$ die Kardinalität, d. h. die Anzahl an Knoten von G und sei u ein Nachbar von v.

Für jedes Molekül der Eingabemenge L berechnet der Algorithmus Graph-Grammatik-Regeln solange der Restgraph $H \subseteq G$ mindestens zwei Knoten aufweist. Dabei verschmilzt er ausgehend vom aktuell betrachteten Knoten $v \subseteq H$ ein Knotenpaar u, v zu uv. Der Algorithmus iteriert über alle Knoten aus V, um alle Regeln zu erzeugen. Beim Erzeugen der Graph-Grammatik-Regeln werden alle möglichen Knotenpaarkombinationen u, v vom initialen Graphen G betrachtet. Um dies zu ermöglichen, ist es notwendig, vor jedem Verschmelzen der Knoten u und v genau $|N(v)|$ Kopien des aktuellen Graphen zu erzeugen, d. h. für jeden Knoten v im Graphen $H \subseteq G$ erzeugt der Algorithmus vor dem Verschmelzen $|N(v)|$ Kopien des aktuellen Graphen. Der Algorithmus iteriert nun über alle Knoten $v \in H$ und speichert für alle Knotenpaare u, v, falls sie noch nicht erzeugt wurden, eine normalisierte Regel ab (Beschreibung zur Erzeugung einer normalisierten Regel siehe Kapitel 5). Hierbei werden iterativ die betrachteten Knotenpaare u, v im Graphen H zu uv verschmolzen. Der Knoten uv wird zum Restgraphen hinzugefügt und die Knoten u und v entfernt. Kanten, die zwischen den Knoten u und v verbunden waren, werden gelöscht. Die übrigen Kanten von u und v bleiben erhalten und behalten ihre Endpunkte. Die erzeugten Restgraphen H werden abgespeichert und die berechneten Regeln ausgegeben. Die Rekursion bricht ab, falls der Restgraph H nur noch einen Knoten aufweist.

8.2 ALGORITHMUS ZUM LERNEN INTERPOLIERENDER GRAPH-GRAMMATIKEN

In Abschnitt 8.1 wurde ein naiver Algorithmus zur Regelerzeugung vorgestellt. Der Algorithmus aus diesem Abschnitt erzeugt Graph-Grammatik-Regeln ähnlich zum naiven aus Abschnitt 8.1. Dabei erhält er die gleichen Eingabedaten. Es werden hier aber nicht alle möglichen Regeln erzeugt, sondern es wird versucht, Regeln mit einer minimalen Labellänge zu erzeugen, die nur die gesuchten Subgraphen erzeugen und gleichzeitig aussagekräftig genug sind, um möglichst

alle negativen Treffer vorab auszuschließen. Somit kann Rechenzeit eingespart werden, weil nicht alle Regeln wie in Abschnitt 8.1 getestet werden müssen, die vielleicht nicht relevant sind. Die Labellänge kann z. B. von der gesuchten Substruktur abhängen, d. h. die Anzahl an Atomen in der gesuchten Substruktur entspricht der Labellänge. Sie kann aber auch frei definiert werden. Beispiel: Wird eine Substruktur gesucht, die drei Atome enthält, müssen nicht Regeln erzeugt werden, die eine Knotenlabellänge von mehr als drei aufweisen, denn die Substruktur sollte bereits mit den erzeugten Regeln aufgefunden werden.

Der Hauptunterschied zu dem Verfahren aus Abschnitt 8.1 besteht darin, dass nicht umgehend alle Regeln, sondern erst in den folgenden Iterationen, falls nötig, berechnet werden. Genauer gesagt, die vorher fest definierte Labellänge gibt an, wie viele Iterationen durchgeführt werden, bis der Algorithmus terminiert. Diese erzeugten Regeln werden anschließend miteinander wie beschrieben und nach ihrer Häufigkeit aufsummiert und sortiert.

8.3 ALGORITHMUS ZUM LERNEN VON GRAPH-GRAMMATIKEN UNTER CHEMISCHEN ASPEKTEN

Der Algorithmus in diesem Abschnitt erzeugt Graph-Grammatik-Regeln analog zum Algorithmus aus Abschnitt 8.2. Der Unterschied hier ist, dass der chemische Aspekt miteinbezogen wird, d. h. es wird nach Atomen oder Verbindungen im Molekül gesucht, die einen geringen Anteil ausmachen (speziell sind) oder eine besondere Bindungsart eingehen. Ein weiterer Unterschied ist, dass nicht alle Möglichkeiten berechnet werden, um eine Regel zu erzeugen und die Labellänge hier beliebig lang werden kann. Denn falls eine spezielle chemische Struktur gefunden wurde, startet der Algorithmus immer an dieser Stelle mit der Regelerzeugung. Es werden hierzu unterschiedliche Ansätze vorgestellt, die miteinander kombiniert werden können.

Die Methode in diesem Abschnitt zur Erzeugung von Graph-Grammatiken durchsucht die Moleküle aus der Lernmenge L vorab nach einer besonderen Struktur, wie z. B. dem Pharmakophor. Das Pharmakophor ist für die pharmakologische Wirkung im Molekül verantwortlich. Eine Auswertung der Zinc-Datenbank für alle All-Purchasable[3] ergab, dass die Bindungszahl im Schnitt weniger als vier aufweist. Moleküle bestehen überwiegend aus den Atomen Stickstoff (N), Sauerstoff (O), Kohlenstoff (C) und Wasserstoff (H) (im Folgenden NOCH-Atome). Diese vier Atome stellen ca. 95% der gesamten Atome in einem Molekül dar. Die Anzahl an kovalenten Bindungen, d. h. die Anzahl an Bindungen zwischen zwei Atomen, liegt

3 **http://zinc.docking.org/subsets/all-purchasable**.

bei unter drei in 99% der untersuchten Moleküle. Die Bindungszahl, d. h. der Knotengrad eines Moleküls, liegt in 96% bei unter vier (siehe Tabelle 5). Aus diesem Grund wird bei der Regelerzeugung diese besondere Struktur ausgenutzt, falls sie in den Molekülen vorhanden ist. Das Ziel ist es, mit möglichst wenigen Regeln die Moleküle der Testmenge richtig zu klassifizieren oder die gewünschten Substrukturen aufzufinden.

Der Algorithmus erhält die gleiche Eingabe wie in Abschnitt 8.2 beschrieben. Für jedes Molekül $G \in L$ erzeugt der Algorithmus die dazugehörigen normalisierten Graph-Grammatik-Regeln rekursiv (Beschreibung siehe Kapitel 5), wobei die Kantenlabel auch hier keine Beachtung finden. Der Algorithmus durchsucht, bevor er mit der Erzeugung von Graph-Grammatik-Regeln beginnt, ein Molekül $G \in L$ auf folgende Eigenschaften und speichert sie für jedes Molekül separat ab. Dabei sollte vorab bekannt sein, dass die Moleküle der Menge L aus der gleichen chemischen Gruppe bestehen. Sollte dies nicht der Fall sein, könnten einzelne erzeugte Regeln nicht auf die gesamte Menge der Moleküle in T verallgemeinert werden.

Die Eigenschaften, die vorab im Molekül durchsucht werden, sind:

(1) Ein nicht N-, O-, C-, H-Atom, das einen Nachbarn aufweist, welches ebenfalls kein N-, O-, C-, H-Atom ist. Diese werden als Knotenpaare abgespeichert.

(2) Einzelne Atome, die kein N-, O-, C-, H-Atom sind.

(3) Atome mit einem Knotengrad > 3.

(4) Anzahl an Kanten zwischen zwei Atomen, d. h. die Anzahl an kovalenten Bindungen > 2.

Der Algorithmus geht wieder analog zur Methode in Abschnitt 8.2 vor. Für jedes Molekül $G \in L$ erzeugt er die dazugehörigen normalisierten Graph-Grammatik-Regeln rekursiv (siehe Kapitel 5). Es werden bei der Erzeugung lediglich die Eigenschaften aus (1), (2), (3), (4) zusätzlich betrachtet. Bei der Erzeugung der Graph-Grammatik-Regeln werden die Eigenschaften in einer Prioritätsreihenfolge abgefragt. Diese erfolgen in der Reihenfolge (1), (2), (3), (4). Die unterschiedlichen Eigenschaften können aber auch kombiniert werden, d. h. besitzt z. B. ein Knotenpaar aus der Menge (1) einen Nachbarknoten, der ebenfalls eine Eigenschaft aus (2), (3), (4) aufweist, wird dieser in den darauf folgenden Iterationen priorisiert zur Regelerzeugung genutzt. Da in dieser Arbeit nur normalisierte Regeln erzeugt werden (siehe Kapitel 5), kann, nachdem ein Label erzeugt wurde, nur ein Element der Mengen (2), (3), (4) gewählt werden. Würde man eine nachfolgende Regel mit einem Element aus (1) nutzen,

verschmilzt man mehr als zwei Knoten gleichzeitig miteinander zur Regelerzeugung. Da auch im Graphen immer nur ein Knotenlabel gleichzeitig vorhanden sein darf, werden immer nur Nachbarn vom aktuellen Knotenlabel, aber alle Möglichkeiten (siehe Abschnitt 8.2), miteinander verschmolzen. Beispiel: Besitzt die Menge aus (1) einen Eintrag, beginnt der Algorithmus bei diesem Knotenpaar (u, v) eine normalisierte Regel zu erzeugen und speichert diese ab. Falls die Menge aus (2) leer sein sollte, überprüft der Algorithmus, ob die Menge aus (3) einen Eintrag besitzt usw. Sei $(u, v) \in V$ ein Knotenpaar aus der Menge (1). Der Algorithmus verschmilzt diese zu uv und erzeugt eine Regel. Nun prüft der Algorithmus, ob ein Nachbarknoten $N(uv)$ vorhanden ist, der Element einer oder mehrerer Mengen aus (2), (3) oder (4) ist. Ist dies der Fall, werden Knoten (uv) und ein Knoten aus (2), (3) oder (4) miteinander verschmolzen, andernfalls wird ein beliebiger Nachbarknoten zur Regelerzeugung gewählt. Eingangs wurde erläutert, dass die Knoteneigenschaften miteinander kombiniert werden können, d. h. falls der Knoten uv z. B. mehrere Nachbarknoten $|N(uv)| \geqslant 2$ aufweisen würde, die gleichzeitig Elemente aus den Mengen (2), (3) und (4) wären, wird der Knoten gewählt, der die meisten zusätzlichen Eigenschaften aufweist. Weisen mehrere Knoten die gleiche Anzahl an zusätzlichen Eigenschaften auf, wird der erste Nachbar ausgewählt. Dies erfolgt auch, falls die Nachbarn keine zusätzlichen Eigenschaften aufweisen. Nach jeder Iteration werden die Elemente in (1), (2), (3), (4) aktualisiert, d. h. dass die bereits zur Regelerzeugung genutzten Knoten aus den Mengen der besonderen Eigenschaften eliminiert werden. Die Rekursion wird wieder so lange ausgeführt, bis jeder Teilgraph $H \subseteq L$ maximal einen Knoten aufweist. Sind nach dem Erzeugen und Anwenden aller möglichen Regeln auf die Menge L nicht alle Moleküle in T richtig zugeordnet, gibt der Algorithmus ebenfalls wieder den prozentualen Anteil der positiven Treffer aus.

8.4 GENERALISIERUNG VON GRAPH-GRAMMATIKEN

In diesem Abschnitt wird eine Methode zur Generalisierung von Graph-Grammatiken beschrieben. Es wird gezeigt, dass durch Generalisierung generell weniger Graph-Grammatik-Regeln benötigt werden, um Moleküle richtig zu klassifizieren oder eine gewünschte Substruktur aufzufinden. In den Abschnitten 8.1, 8.2 und 8.3 wurden die erzeugten Regeln aller Moleküle der Eingabemenge gemeinsam in einer Menge abgespeichert und nach einer Häufigkeitsverteilung sortiert und ein gewisser Prozentsatz zur Regelmenge hinzugefügt. Zur Generalisierung werden die Regeln nun für jedes Molekül der Eingabemenge einzeln in einem gesonderten Set zwischengespeichert. Diese erzeugten Regelsets werden nun miteinander verglichen sowie

Tabelle 5: Verteilung – Zinc Datenbank der All-Purchasable
Es wurden 22 Millionen Moleküle untersucht, dabei wurde die Anzahl an Elementen, die Bindungszahl, die kovalenten Bindungen (alle aufgezählten Hydrogenatome sind explizit geschrieben, die tatsächliche Anzahl liegt wesentlich höher) aufsummiert. Die Tabelle gibt die jeweiligen Verteilung an.

Element	Anzahl	Anteil	Kovalente Bindungen	Anzahl	Anteil	Bindungszahl	Anzahl	Anteil
C	312.374.294	71%	1	347.049.551	73%	1	88.020.220	20%
N	48.487.451	11%	2	124.890.722	26%	2	214.754.914	49%
O	44.619.904	10%	3	948.242	1%	3	117.703.511	27%
H*	16.193.861	3%				4	18.784.046	4%
S	7.825.885	2%				5	35	0%
F	5.235.224	1%						
Cl	3.494.122	1%						
Br	923.976	0%						
I	82.527	0%						
P	25.500	0%						

deren gemeinsame Regeln ausgegeben und zur Graph-Grammatik zur Klassifikation oder Substruktursuche hinzugefügt. Zur Generalisierung können die Knotengrade, falls nötig, hier auch wieder vernachlässigt werden.

Genauer, die Regeln, die durch die Algorithmen aus den Abschnitten 8.1 und 8.2 erzeugt wurden, werden hier separat für jedes Molekül jeweils in einem Set abgespeichert. Nachdem alle Regeln berechnet worden sind, werden die Regelsets nach Gemeinsamkeiten untersucht. Die erzeugten Sets können in mehreren Kombinationen miteinander verglichen werden, d. h. wurden z. B. fünf Regelsets erzeugt, weil fünf Moleküle in der Lernmenge zur Regelerzeugung genutzt wurden, werden zuerst alle fünf erzeugten Sets miteinander verglichen und deren gemeinsame Regeln ausgegeben. Liefert der Vergleich Treffer, werden diese zur Graph-Grammatik hinzugefügt. Andernfalls werden $k-1$ bis $k=2$ Sets aus den fünf ohne Wiederholung und Berücksichtigung der Reihenfolge ausgewählt und miteinander verglichen, d. h. es werden alle Vierer-Kombinationen usw. miteinander verglichen, danach alle Dreierkombination usw. bis mindestens eine gemeinsame Regel ausgegeben wurde, oder alle Kombinationen miteinander verglichen wurden. Einerkombinationen werden natürlich nicht betrachtet, da in diesem Fall alle erzeugten Regeln aus den Sets zur Graph-Grammatik hinzugefügt werden würden.

Beispiel: Es wurden die unten aufgeführten Regelmengen berechnet:

$$1 = \{A, B, C\},\ 2 = \{A, C, D\},\ 3 = \{D, E, F\},\ 4 = \{H, I, J\}$$

Es werden nun alle vier miteinander verglichen. Weisen sie mindestens eine gemeinsame Regel auf, werden diese ausgegeben. Im obigen

Fall ist es offensichtlich nicht möglich, eine gemeinsame Regel in allen vier Teilmengen auszumachen. Aus diesem Grund werden jetzt alle Kombinationen an Dreiermengen miteinander verglichen, d. h. $k - 1 = 3$. Auch in diesem Fall werden keine gemeinsamen Regeln aufgefunden. Nun iteriert der Algorithmus über alle Zweierkombinationen. Dies liefert für den Vergleich der Teilmengen 1 und 2 das Ergebnis $\{A, C\}$ und $\{D\}$ für die Mengen 2 und 3. Nun wird das Set mit den meisten gemeinsamen gefundenen Elementen ausgewählt. Falls zwei resultierende Mengen die gleiche Anzahl an Regeln aufweisen, wird das erste Set ausgewählt und zur Graph-Grammatik hinzugefügt. Der Vergleich der Regelmengen bricht ab, sobald eine Menge mit mehr als einer gemeinsamen Regel gefunden wurde oder kein Vergleich zu einem Ergebnis geführt hat.

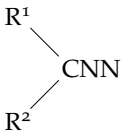
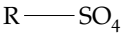
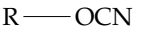
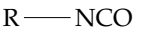
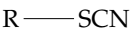
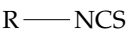
8.5 DATEN UND HYPERPARAMETER

Mit über 35 Millionen molekularen Strukturen ist die Zinc-Datenbank eine der größten freien Datenbanken, die kommerziell Strukturverbindungen für das virtuelle Screening anbietet. Sie wird von der Universität Kalifornien in San Francisco (USA) bereitgestellt [76]. Strukturen werden in unterschiedlichen Formaten wie z. B. SMILES angeboten. Das SMILES-Format wurde in Abschnitt 2.1.1 beschrieben.

Die Daten der Lern- und Testmengen sind zur Berechnung der Ergebnisse wie folgt aufgeteilt: Die Moleküle der Lernmenge gehören entweder der gewünschten funktionalen Gruppe an oder sie weisen alle eine gemeinsame Substruktur auf. Die untersuchten funktionellen Klassen sind: Hydrazone, Sulfate, Cyanate, Isocyanate, Thiocyanate und Isothiocyanate. Cyanate sind Salze, die ausgewählt wurden, um die erzielten Ergebnisse bei ähnlichen Strukturen miteinander vergleichen zu können. Die anderen Strukturklassen wurden zufällig ausgewählt. Die allgemeine Strukturformel der untersuchten funktionalen Gruppen, die beim Vergleich genutzt wurden, sind in Tabelle 6 aufgeführt. Die erste Spalte gibt die jeweilige funktionale Gruppe (Stoffklasse) an, die zweite Spalte die Molekülstruktur. Das R stellt in der Chemie einen organischen Rest dar. Die Hydrazonegruppe besitzt zwei Restgruppen R^1 und R^2.

Für jede funktionelle Klasse wurden für die Lernmenge genau 20 Moleküle ausgewählt, d. h. aus allen zur Verfügung stehenden Molekülen pro funktionaler Klasse wurden die Moleküle nach der Länge ihrer Atome sortiert. Es wurden 20 derjenigen ausgewählt, die die wenigsten Atome aufwiesen, um den Bias für jede Klasse so gering wie möglich zu halten, d. h., dass sie im besten Fall nur die gesuchte Substruktur aufzeigten. Es wurden Testmengen unterschiedlicher Größe erstellt. Die erste Testmenge bestand aus sechs unterschiedlichen funktionalen Gruppen plus 40 Moleküle, die keiner dieser sechs

Tabelle 6: Überblick der genutzten funktionellen Gruppen beim Vergleich der Ansätze, Graph-Grammatik – maschinelles Lernen
Das R stellt einen organischen Rest im Molekül dar [2].

Stoffklasse	Struktur
Hydrazone	
Sulfate	
Cyanate	
Isocyanate	
Thiocyanate	
Isothiocyanate	

Gruppen angehören. Sie dienten als zusätzliche Testelemente. Die erste Testmenge bestand aus insgesamt 160 Molekülen, die zweite aus 750 Molekülen für jede der sechs zu untersuchenden Mengen plus zusätzlich 500 Moleküle, die ebenfalls zu keiner dieser sechs Gruppen angehören, d. h. es wurden insgesamt 5.000 Moleküle betrachtet.

Um die Güte der erzeugten Graph-Grammatik-Regeln, die durch die Lernalgorithmen aus den Abschnitten 8.2 und 8.3 berechnet worden sind, zu überprüfen, wird dem Algorithmus aus Kapitel 6 eine bestimmte Anzahl an Graph-Grammatik-Regeln zur Substruktursuche oder Klassifikation übergeben. Die Güte gibt hier die prozentuale Menge an positiven Treffern an, die vom Algorithmus aus Kapitel 6 erkannt wurden. Für jeden der vorgestellten Ansätze werden unterschiedlich viele Regeln berechnet und genutzt. Nachfolgend wird erläutert, wie diese für den jeweiligen Algorithmus bestimmt und welche davon eingesetzt wurden. Zusätzlich wird ähnlich wie beim maschinellen Lernen ein Hyperparameter adaptiv an die Daten angepasst, um ferner die Such- oder Klassifizierungsergebnisse zu optimieren. Dieser ist die maximal zu berechnende Labellänge. Wurde z. B. ein Sulfat gesucht, das in der Regel ein Schwefel- und vier Sauerstoff-Atome aufweist, wurde anfangs ein Wert von fünf gewählt (Anzahl der Atome). Wurden mit diesem Wert gute Ergebnisse zur Substruktursuche erzielt, ist dieser Wert übernommen worden, andernfalls wurde er so lange angepasst, bis ausreichend gute Ergebnis erzielt wurden, ähnlich wie beim überwachten Lernen (siehe Abschnitt 2.5.2.4).

Beim Algorithmus aus Abschnitt 8.2 wird ohne den Einsatz des Generalisierungsalgorithmus aus Abschnitt 8.4 und der Knotengradvernachlässigung ein vorher frei definierter Prozentsatz an erzeugten

Regeln als Eingabemenge an den Algorithmus aus Kapitel 6 übergeben. Dabei werden alle erzeugten Regeln der Lernmenge nach ihrer Häufigkeit aufsummiert und eine bestimmte Menge ausgewählt und zur Substruktursuche / Klassifikation genutzt. Es wurden unterschiedlich viele Regeln für jede Testinstanz getestet. Am Anfang wurden fast alle, die generiert wurden, genutzt. Sukzessive wurde diese Anzahl aber adaptiv angepasst. Es hatte sich herausgestellt, dass in der Regel nicht mehr als 30 Regeln nötig waren, um bereits gute Ergebnisse zu erzielen. Aus diesem Grund wurden bei der Berechnung der Ergebnisse aus Tabelle 7 genau 30 Regeln zur Substruktursuche eingesetzt.

Falls der Generalisierungsalgorithmus zum Einsatz kam, wurde ebenso der Hyperparameter Labellänge wie beschrieben angepasst. Nur die Auswahl der Regeln erfolgte wie in Abschnitt 8.4 definiert. Wurde zusätzlich der Knotengrad (Tabelle 9) vernachlässigt, werden beim Einsatz des Algorithmus aus Abschnitt 8.2 die gleichen Regeln, die auch durch den Generalisierungsalgorithmus berechnet wurden, genutzt. Auch blieb der Hyperparameter Labellänge unverändert.

Die erzeugten Graph-Grammatik-Regeln des Algorithmus aus Abschnitt 8.3 werden wieder wie beschrieben in einer einzelnen Menge gespeichert und nach ihrer Häufigkeit sortiert. Es wurden im Allgemeinen 10% der häufigsten Regeln ausgewählt und dem Algorithmus aus Kapitel 6 zur Suche übergeben. Dieser Wert konnte aber auch leicht nach unten oder oben abweichen, je nachdem wie viele Regeln generiert worden sind. Aber auch hier wurde der Hyperparameter Labellänge adaptiv an die jeweilige funktionale Gruppe, wie oben beschrieben, angepasst.

8.6 ERGEBNISSE

Die Ergebnisse, die mit den unterschiedlichen Ansätzen aus dem Kapitel 8 erzielt worden sind, werden nachfolgend vorgestellt und anschließend in einer Tabelle zusammengefasst und bewertet.

Die Ergebnisse werden in vier Mengen unterteilt, d. h. True-Positives, True-Negatives, False-Positives und False-Negatives. In der Menge True-Positives sind die Moleküle enthalten, die anhand der jeweils erzeugten Regeln Moleküle richtig klassifiziert/aufgefunden hat, d. h. auch zur gesuchten Gruppe von Molekülen gehören. Die Menge der True-Negatives weist Moleküle auf, die nicht zur Menge gehören und auch so eingeordnet wurden. Die Menge an False-Positives enthält Elemente, die fälschlicherweise der gesuchten Menge zugeordnet worden sind. Die Menge der False-Negatives beinhaltet Moleküle, die zur gesuchten Menge gehören, aber nicht als solche

vom Algorithmus erkannt wurden. Diese vier Werte werden in Prozent ausgegeben. Dabei werden die Wertegruppen True-Positives / False-Negatives und True-Negatives / False-Positives zu jeweils 100% aufsummiert.

Die Ergebnistabellen sind wie folgt aufgebaut: Die erste Spalte gibt die gesuchte chemische funktionale Gruppe an, die zweite den prozentualen Anteil der True-Positives (TP), d. h. wurden alle Moleküle der Testmenge richtig zugeordnet, weist der True-Positive einen Wert von 100% auf. Wurden z. B. nur 80% Moleküle richtig zugeordnet, weist der False-Negative einen Wert von 20% auf. Die dritte Spalte gibt den prozentualen Anteil der True-Negatives (TN) an. Das sind die Moleküle, die nicht zur jeweiligen gesuchten Gruppe gehören und auch so klassifiziert wurden. Wies der True-Negative einen Wert von z. B. 60% auf, sollte der dazugehörige False-Positive-Wert einen Wert von 40% aufweisen. Die vierte Spalte gibt den prozentualen Anteil an Molekülen an, die fälschlicherweise nicht als Element der gesuchten Klasse aufgefunden wurden und somit nicht genügend Regeln erzeugt worden sind, die diese Elemente zur jeweiligen Gruppe matchen. Diese werden mit False-Negatives (FN) angegeben. Die fünfte Spalte zeigt den prozentualen Anteil an Molekülen an, die falsch zu einer Gruppe zugeordnet worden sind, d. h., dass die vorhandenen Regeln auch Moleküle anderer Gruppen matchen und somit nicht eine ausreichend einschränkende Wirkung aufweisen. Diese werden in der Tabelle als False-Positives (FP) ausgewiesen. Die sechste Spalte gibt die Anzahl an Regeln an, die für die jeweilige Klasse im respektiven Algorithmus eingesetzt worden sind. Dies soll später aufzeigen, dass durch Generalisierung weniger Regeln zur Suche benötigt werden als ohne. Die siebte Spalte zeigt auf, bis zu welcher Labellänge (Label) der Algorithmus die Berechnungen durchgeführt hat, um eine Substruktur ausfindig zu machen, d. h. die Labellänge sagt aus, wie viele Knoten maximal zusammengezogen werden. Die achte Spalte beschreibt, ob die Knotengrade (Grad) berücksichtigt worden sind. Falls „ja“ in der jeweiligen Spalte notiert ist, wurden die Knotengrade nicht betrachtet. Die neunte Spalte zeigt, ob die Regeln unter Zuhilfenahme des Generalisierungsalgorithmus (General) aus Abschnitt 8.4 erstellt worden sind.

In Tabelle 7 sind die Ergebnisse festgehalten, die mit den erzeugten Regeln aus dem Algorithmus aus Abschnitt 8.2 berechnet worden sind. Zur Substruktursuche sind die jeweils 30 häufigsten Regeln zur Graph-Grammatik hinzugefügt und anschließend mit dem Algorithmus aus Kapitel 6 die Substruktursuche berechnet worden. Die Testmenge umfasste hier 160 Moleküle. Mit den erzeugten Regeln wurden generell zu viele False-Positives erzeugt. Dennoch konnten alle Elemente der Sulfatgruppe richtig berechnet werden. Auch über 85% der Thiocyanattestgruppe konnten richtig aufgefunden werden. Die

Tabelle 7: Ergebnisse – Algorithmus zum Lernen von Interpolierenden Graph-Grammatiken bei einer Datenmenge von 160 Molekülen

Gruppe	TP	TN	FN	FP	Regeln	Label	Grad	General
Hydrazone	60,00	73,55	40,00	26,45	30	4	ohne	ohne
Sulfate	100,00	53,55	0,00	46,65	30	4	ohne	ohne
Cyanate	40,00	41,85	60,00	52,15	30	4	ohne	ohne
Isocyanate	70,00	47,83	30,00	19,27	30	4	ohne	ohne
Thiocyanate	85,14	51,43	14,88	48,57	30	5	ohne	ohne
Isothiocyanate	70,00	50,71	30,00	49,29	30	4	ohne	ohne

Tabelle 8: Ergebnisse – Kombination Algorithmus Interpolierender Graph-Grammatik und Generalisierungsalgorithmus, 160 Moleküle

Gruppe	TP	TN	FN	FP	Regeln	Label	Grad	General
Hydrazone	100,00	97,86	0,00	2,14	2	2	ohne	ja
Sulfate	100,00	100,00	0,00	0,00	13	4	ohne	ja
Cyanate	100,00	100,00	0,00	0,00	8	3	ohne	ja
Isocyanate	100,00	87,14	0,00	12,86	17	4	ohne	ja
Thiocyanate	100,00	100,00	0,00	0,00	8	3	ohne	ja
Isothiocyanate	100,00	100,00	0,00	0,00	8	3	ohne	ja

Cyanattestgruppe konnte lediglich zu 40% richtig klassifiziert werden.

In Tabelle 8 wurde zusätzlich zum Algorithmus aus Abschnitt 8.2 auch der Generalisierungsalgorithmus aus Abschnitt 8.4 zur Regelerzeugung eingesetzt. Auch hier enthielt die Lern- und Testmenge 20 pro Testgruppe und 160 Moleküle. Wie man der Tabelle entnehmen kann, wurde durch den Einsatz des Algorithmus aus Abschnitt 8.4 der False-Negative-Anteil bei allen Gruppen auf 0% reduziert. Alle Elemente der Testmengen konnten zu 100% richtig zugeordnet werden. Lediglich die Testmenge der Isocyanate wies einen False-Positive-Wert von 12,86% auf. Wurden in den Ergebnissen aus Tabelle 7 noch 30 Regeln zur Substruktursuche eingesetzt, konnten jetzt mit nur zwei fast alle Elemente der Testmenge der Hydrazone richtig zugeordnet werden. Es waren generell nur Berechnungen bis zu einer Labellänge von maximal vier nötig, um eine Genauigkeit von 100% zu erreichen. Dennoch wiesen auch unter Zuhilfenahme des Generalisierungsalgorithmus die Isocyanate einen False-Positive-Anteil von über 12% auf (siehe Tabelle 8).

Tabelle 9: Ergebnisse – Kombination Algorithmus Interpolierender Graph-Grammatik, Generalisierungsalgorithmus und Knotengradvernachlässigung, 160 Moleküle

Gruppe	TP	TN	FN	FP	Regeln	Label	Grad	General
Hydrazone	100,00	97,86	0,00	2,14	2	2	ja	ja
Sulfate	100,00	100,00	0,00	0,00	13	4	ja	ja
Cyanate	100,00	62,14	0,00	37,86	8	3	ja	ja
Isocyanate	100,00	60,72	0,00	39,28	17	4	ja	ja
Thiocyanate	100,00	80,72	0,00	19,28	8	3	ja	ja
Isothiocyanate	100,00	80,72	0,00	19,28	8	3	ja	ja

In Tabelle 9 wurden zusätzlich zum Algorithmus aus Abschnitt 8.2 und dem Generalisierungsalgorithmus aus Abschnitt 8.4 auch die Knotengrade bei der Regelerzeugung vernachlässigt, denn es kann vorkommen, dass eine allgemeine Substruktur erkannt, aber unterschiedliche Regeln mit diversen Knotengraden zu dieser erzeugt wurden. Um dieses Problem zu umgehen und zwecks der Generalisierung muss überprüft werden, ob mehrere Regeln berechnet wurden, die die gleichen Knotenlabel miteinander verschmelzen würden. Ist dies der Fall, sollte die Überprüfung der Knotengrade in den erzeugten Regeln vernachlässigt werden, d. h. vor jeder möglichen Verschmelzung der Subgraphen H_i und H_j überprüft der Algorithmus hier nur, ob mindestens eine Regel existiert, dass die Knotenlabel der Subgraphen H_i und H_j kompatibel sind. Die zwei zu verschmelzenden Komponenten müssen lediglich die richtige Anzahl an Verbindungskanten zueinander aufweisen, d. h. die Kantenzahl aus der Menge $\delta(u) \triangle \delta(v)$ kann vernachlässigt werden. Somit sollte es möglich sein, mit weniger Regeln die gleichen Substrukturen aufzufinden und eine ähnliche Ausdruckskraft zu erreichen.

Die Algorithmen erhielten hier die gleichen Inputdaten wie in den Beispielen zuvor. Auch die Berechnung der Labellänge blieb gleich. Wie man Tabelle 9 entnehmen kann, wurden zu viele Treffer durch die Vernachlässigung der Knotengrade erzeugt, d. h. Moleküle wurden fälschlicherweise einer fremden Klasse zugeordnet (siehe FP-Werte). Die True-Positive-Werte blieben, wie erwartet, konstant bei 100%. Es wurden die Regeln unter Zuhilfenahme des Generalisierungsalgorithmus aus Abschnitt 8.4 erzeugt und angewendet. Zusätzlich wurden die Knotengrade zum Zwecke der Generalisierung vernachlässigt. Die Lernmenge bestand aus jeweils 20 Elementen pro Gruppe. Die Testmenge bestand aus 160 Molekülen.

In Tabelle 10 wurden die gleichen Verfahren eingesetzt wie in Tabelle 8. Die Testmengen enthielten hier aber jeweils 750 Moleküle pro

Tabelle 10: Ergebnisse – Kombination Algorithmus Interpolierender Graph-Grammatik und Generalisierungsalgorithmus, 5.000 Moleküle

Gruppe	TP	TN	FN	FP	Regeln	Label	Grad	General
Hydrazone	99,86	94,02	0,14	5,97	2	2	ohne	ja
Sulfate	99,87	99,95	0,13	0,05	13	4	ohne	ja
Cyanate	99,84	99,88	0,16	0,12	8	3	ohne	ja
Isocyanate	99,75	97,05	0,25	2,95	17	4	ohne	ja
Thiocyanate	99,73	99,87	0,27	0,13	8	3	ohne	ja
Isothiocyanate	99,48	99,15	0,52	0,85	8	3	ohne	ja

Gruppe. Fast alle Testmengen konnten einen True-Postive-Wert von über 99% erzielen. Die gesamte Tabelle wies im Verhältnis ähnliche Ergebnisse auf wie Tabelle 8, d. h. dass die erzielten Ergebnisse aus Tabelle 8 bestätigt wurden und die berechneten Regeln hinreichend sind, um diese Molekülgruppen für die Testdaten richtig zu klassifizieren oder eine gemeinsame Substruktur aufzufinden.

Die folgenden Ergebnisse wurden mit dem Algorithmus aus Abschnitt 8.3 erzeugt. Der Algorithmus bekam auch hier die gleichen Eingabedaten wie in Tabelle 8 zum Erzeugen der Graph-Grammatik-Regeln. Die Lernmenge bestand jeweils aus 20 Molekülen pro Gruppe und die Testmengen aus 160 Molekülen. Bevor der Algorithmus aus Abschnitt 8.3 mit der Regelerzeugung beginnt, durchsucht er Moleküle vorab auf die beschriebene besondere Struktur. Bei der späteren Regelerzeugung beginnt er oftmals an der gleichen Stelle im Graphen. Aus diesem Grund war es nicht möglich, mit dem Generalisierungsalgorithmus ausreichend Treffer zu erzeugen. Somit wurden die häufigsten Regeln zufällig manuell ausgewählt, weil für jede Gruppe unterschiedlich viele generiert wurden. Die Anzahl an Regeln, die für jede Gruppe genutzt wurden, können Tabelle 11 entnommen werden. Mit den erzeugten Regeln konnten ebenfalls gute bis sehr gute Ergebnisse bei der Substruktursuche erzielt werden. Die Elemente der Testmengen der Sulfate und Isothiocyanate konnten bis zu einem geringen Prozentsatz richtig zugeordnet werden. Die Anzahl an Regeln, die erzeugt wurden für jede Gruppe, sind kongruent zu denen, die mit dem Algorithmus aus Abschnitt 8.2 erzeugt wurden.

In Tabelle 12 wurde zum Generieren der Ergebnisse der Algorithmus aus Abschnitt 8.3 genutzt. Zusätzlich wurden die Knotengrade vernachlässigt. Die Lernmenge bestand aus jeweils 20 Elementen pro Gruppe, die Testmenge aus 160 Molekülen. Somit konnte der False-Negative-Anteil auch hier in fast allen Gruppen auf unter 15%, außer in der Cyanattestgruppe, verringert werden. Im Fall der Sulfat-,

Tabelle 11: Ergebnisse – Algorithmus unter chemischen Aspekten, 160 Moleküle

Gruppe	TP	TN	FN	FP	Regeln	Label	Grad	General
Hydrazone	55,00	92,15	45,00	7,85	6	6	ohne	ohne
Sulfate	100,00	100,00	0,00	0,00	4	5	ohne	ohne
Cyanate	50,00	82,15	50,00	17,85	18	5	ohne	ohne
Isocyanate	85,00	95,00	15,00	5,00	16	5	ohne	ohne
Thiocyanate	40,00	90,00	60,00	10,00	6	3	ohne	ohne
Isothiocyanate	80,00	100,00	20,00	0,00	7	6	ohne	ohne

Tabelle 12: Ergebnisse – Algorithmus unter chemischen Aspekten und zusätzlicher Knotengradvernachlässigung, 160 Moleküle

Gruppe	TP	TN	FN	FP	Regeln	Label	Grad	General
Hydrazone	100,00	86,42	0,00	13,58	6	6	ja	ohne
Sulfate	100,00	100,00	0,00	0,00	4	5	ja	ohne
Cyanate	80,00	47,85	20,00	52,15	18	5	ja	ohne
Isocyanate	100,00	59,29	0,00	40,71	16	5	ja	ohne
Thiocyanate	85,04	86,42	14,96	13,58	6	3	ja	ohne
Isothiocyanate	95,04	85,00	4,96	15,00	7	6	ja	ohne

Hydrazone- und Isothiocyanattestgruppe konnten fast alle Elemente der Testmoleküle richtig zur jeweiligen Gruppe zugeordnet werden. Im Fall der Isocyanat- und Hydrazonegruppe konnten durch die Knotengradvernachlässigung alle Moleküle richtig aufgefunden werden (siehe Tabelle 12).

Auf die Berechnung der Ergebnisse unter dem chemischen Aspekt wurde beim Ansatz aus Abschnitt 8.3 bei den Testmengen für die jeweils 750 Moleküle aufgrund der etwas schlechteren Ergebnisse im Vergleich zur Tabelle 10 verzichtet. Lediglich die Ergebnisse der Testmenge der Sulfate wurden generiert, weil diese ein nicht N, O, C, H-Atom, das Schwefelatom, aufweisen. Diese sind in der Tabelle 13 festgehalten. Wie man erkennen kann, wurde auch hier ein True-Positive-Wert von fast 100% erreicht.

Die vorgestellten Algorithmen zur Generierung von Graph-Grammatik-Regeln haben sich in den Testszenarien als nutzbar zur Substruktursuche oder Klassifikation erwiesen. Der Algorithmus aus Abschnitt 8.2 konnte bereits bei einer festen Anzahl an Regeln und einer kleinen Lernmenge von 20 Molekülen gute Zuordnungswer-

Tabelle 13: Ergebnisse – Algorithmus unter chemischen Aspekten, 5.000 Moleküle

Gruppe	TP	TN	FN	FP	Regeln	Label	Grad	General
Sulfate	99,94	99,73	0,06	0,27	4	5	ohne	ohne

te erreichen. Hierbei konnten True-Positive-Werte von 100% erreicht werden. Durch den zusätzlichen Einsatz des Generalisierungsalgorithmus aus Abschnitt 8.4 konnten fast alle Zuordnungswerte deutlich verbessert werden. In allen Gruppen fiel der False-Negative-Wert hier auf annähernd 0%. Auch der False-Positive-Wert fiel rapide ab. Somit konnten fast alle Testgruppen zu 100% richtig zugeordnet werden. Beim Testen der Hydrazonetestgruppe konnte mit dem Generalisierungsalgorithmus ein Wert von 100% erreicht werden. Hierbei mussten nur zwei Regeln genutzt werden.

Bei Vernachlässigung der Knotengrade konnten fast immer alle 20 Moleküle der jeweiligen Testgruppe richtig zugeordnet werden. Dabei stieg aber in einem Fall der False-Positive von 0% auf über 33%, wie z. B. bei den Cyanaten, an. Der Wert konnte aber auch konstant bei 0%, wie z. B. bei den Sulfaten, gehalten werden. Im Allgemeinen ist festzuhalten, dass die Vernachlässigung der Knotengrade helfen kann, Substrukturen einer Gruppe zuzuordnen, aber dadurch oft sehr viele False-Positive-Treffer erzeugt werden.

Bei den Testmengen mit 5.000 Molekülen unter Einsatz der Algorithmen aus den Abschnitten 8.2 und 8.4 konnten ebenfalls gute bis sehr gute Werte berechnet werden. Die Werte, die erreicht wurden, sind kongruent zu den Testgruppen der 160 Moleküle. Der False-Positive-Anteil wies hier einmalig einen Wert von ca. 6% in der Testgruppe für Hydrazone auf.

Mit dem Algorithmus aus Abschnitt 8.3, der unter Einbeziehung von chemischen Aspekten Regeln generierte, konnten ebenfalls gute bis sehr gute Werte bei der Substruktursuche erzielt werden. Die besten True-Positive-Werte lagen bei 100% in der Testgruppe für Sulfate und bei 80% bei den Isothiocyanaten. Bei den Sulfaten konnte dieser Wert erreicht werden, weil die spezielle Struktur, die ein Sulfat aufweist (Schwefelatom), genutzt wurde. Wurden die Knotengrade im Algorithmus vernachlässigt, verbesserte sich der True-Positive-Anteil für fast alle Werte. Lediglich der False-Positive-Anteil verschlechterte sich in einigen Fällen. Beim Versuch, 5.000 Moleküle der Testmenge für Sulfate unter Vernachlässigung der Knotengrade richtig zuzuordnen, konnte ein True-Positive-Anteil von 99, 94% erreicht werden.

Durch den Generalisierungsalgorithmus konnte gezeigt werden, dass wesentlich weniger Regeln benötigt werden, um eine Testgruppe richtig zuzuordnen (siehe Hydrazone in Tabelle 8). In fast allen Fällen erzeugte er qualitativ bessere Graph-Grammatik-Regeln zur Substruktursuche / Klassifikation als die anderen hier vorgestellten.

9 KLASSIFIZIERUNG VON MOLEKÜLEN MITTELS MASCHINELLEN LERNENS

Ziel dieses Kapitels ist es, Moleküle unter Zuhilfenahme des maschinellen Lernens, genauer mittels der Tensorflow-API zu klassifizieren. Moleküle werden unter Verwendung eines rekurrenten neuronalen Netzes und des Random Forest Verfahrens (siehe Abschnitte 2.5.2 und 2.5.3) ihren chemischen funktionalen Gruppen zugeordnet. TensorFlow ist eine Open Source-Software-Bibliothek zur datenstromorientierten Programmierung und findet Anwendung im Bereich des maschinellen Lernens. Sie wird seit 2015 vom Google Brain Team entwickelt und ist unter anderem für die Programmiersprachen C++ und `Python` verfügbar. In TensorFlow werden mathematische Operationen in Form von Graphen dargestellt. TensorFlow ist flexibel einsetzbar und kann auf die unterschiedlichsten Problemstellungen angewendet werden. Google nutzt TensorFlow in kommerziellen Produkten, wie z. B. der Spracherkennung, Gmail, Google Fotos und der Google-Suche [5]. In Abschnitt 9.1 folgt die Klassifizierung unter Einsatz von RNNs und in Abschnitt 9.2 mittels des Random Forest Ansatzes.

9.1 KLASSIFIZIERUNG MITTELS REKURRENTEN NEURONALEN NETZWERKEN

Zur Klassifizierung wird ein einschichtiges rekurrentes neuronales Netzwerk genutzt, das durch LSTM-Zellen implementiert wird. Die Daten stammen aus der Zinc-Datenbank[1] und werden in eine Lern- und Testmenge unterteilt. Die Aufteilung erfolgt in circa 75% Lern- und 25% Testdaten. Dabei werden die vorliegenden SMILES aufgrund ihrer Mehrdeutigkeit in unique SMILES umgewandelt und anschließend mittels des Sequenz-zu-Sequenz-Modells (seq2seq) in eine von Tensorflow nutzbare Sprache übersetzt. Das seq2seq-Modell besteht aus zwei Hauptteilen, einem Kodierer und einem Dekodierer, die ein Paar bilden und jeweils aus einer RNN-Schicht bestehen. Der Kodierer liest einen Ausgangssatz ein und kodiert ihn in einen Vektor fester Länge. Der Dekodierer gibt anschließend eine Übersetzung des kodierten Vektors aus. Das gesamte System, das aus Kodierer und Dekodierer für ein Sprachpaar besteht, wird gemeinsam trainiert, um die Wahrscheinlichkeit einer korrekten Übersetzung bei einem Quellsatz zu maximieren [6, 22].

1 **http://zinc.docking.org/subsets/all-purchasable**.

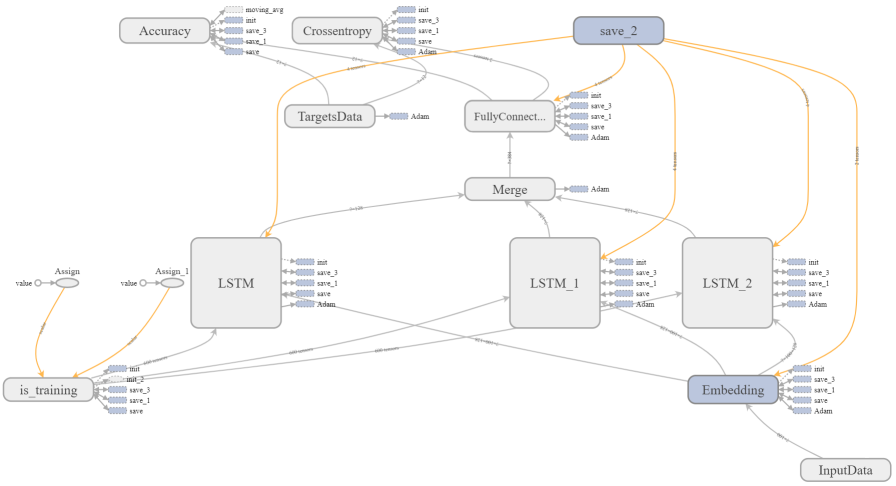


Abbildung 45: Aufbau RNN-Netz

Das zur Optimierung eingesetzte RNN ist vollständig vernetzt. Es besteht aus drei LSTM Zellen (LSTM, $LSTM_1$ und $LSTM_2$). In der Abbildung wurde der Optimierer Adam eingesetzt [5].

9.1.1 *Aufbau des Rekurrenten Neuronalen Netzwerks*

Die zur Optimierung eingesetzten RNNs sind vollständig vernetzt. Sie bestehen aus drei LSTM Zellen. LSTM, $LSTM_1$, $LSTM_2$ und weisen jeweils eine Größe von 128 auf. Das erste Netzwerk, das eingesetzt wird, führt den Output der LSTM-Schichten zusammen (siehe Abbildung 45). Als Optimierer werden die am häufigsten genutzten Adagrad, RMSprop, Adadelta und Adam eingesetzt [111]. Als Aktivierungsfunktionen kommen ReLu und Softmax zum Einsatz, da sich diese in der Vergangenheit als effizient bei der Klassifizierung von Daten herausstellten [126]. Da die Moleküle in Klassen eingeteilt werden sollen, wird ebenso die Fehlerfunktion Crossentropy genutzt. Des Weiteren wird die L2- Regularisierungstechnik verwendet. Anschließend werden die Ausgabewerte miteinander verglichen. Die Daten werden unterschiedlich lang trainiert (Anzahl Epochen). Die Eingabedimension des Netzwerks beträgt 10.000, die Ausgabedimension 128 und die Batchgröße 32.

9.1.2 *Daten und Hyperparameter*

Das Preprocessing der Daten erfolgt mit dem seq2seq-Modell, d. h. die unique SMILES werden zur Nutzung in TensorFlow unter Zuhilfenahme von seq2seq umgewandelt. Das Datenmaterial zur Klassifizierung weist unterschiedliche Datengrößen auf. Die Lern- und Testmengen bestehen aus Datenpaaren der Größe $(100,\ 25)$, $(8.000,\ 2.000)$

und (500.000, 100.000). Dabei wurden je nach Verfügbarkeit diverse funktionale Klassen erstellt. Die Anzahl an Klassen liegt zwischen vier (sie bestanden aus den Gruppen Nitrile, Ketone, Nitro, Carbonsäureamide) und sechs (diese enthielten zusätzlich die Gruppen Oxime und konjugierte Ester).

Aufgrund der Vielzahl an zu definierenden Hyperparametern beim RNN-Ansatz wird sich in dieser Arbeit auf die wichtigsten beschränkt. Hyperparameter stellen die Werte dar, die vor dem Lernprozess im Netzwerk definiert werden und an das jeweilige Problem angepasst werden müssen. Die Hyperparameter, die in den RNN-Modellen adaptiv an die Daten angepasst wurden, sind die Dropout- und Lernrate und die Anzahl an zu trainierenden Epochen.

9.1.3 *Ergebnisse*

Als erstes werden Ergebnisse mit kleinen Datenmengen zur Klassifizierung vorgestellt. Die funktionellen Klassen, die zur Auswertung der Ergebnisse der Klassen (100, 25) und (8.000, 2.000) genutzt werden, sind Carbonsäureamide, Ketone, Nitrile, Oxime, C-Nitro und Ester. Diese Daten wurden 600 beziehungsweise 50 Epochen trainiert. Abschließend werden die Ergebnisse mit den großen Datenmengen vorgestellt. Die Daten bestehen aus den Klassen Carbonsäure, Organosulfate, Peroxide und Peroxy. Die Datensätze für die Datenklassen (500.000, 100.000) wurden nach ihrer Verfügbarkeit ausgewählt, d. h., dass diese genannten Klassen Datenmengen größer als 500.000 Moleküle aufwiesen. Diese Daten wurden eine und zehn Epochen lang trainiert. Die anfängliche Lernrate betrug in allen Fällen 0,001. Sie wurde aber sukzessive angepasst. Alle Ergebnisse wurden gerundet, die Genauigkeits- und Validierungswerte auf ganze Zahlen und die Hyperparameter auf die erste Kommastelle, die ungleich null ist.

In Tabelle 14 wird ein Auszug der erzielten Ergebnisse des Datensatzes (100, 25) unter Verwendung des RNN-Netzes aus Abbildung 45 vorgestellt. Die dritte Spalte weist hierbei die Lernrate in Prozent auf. Die vierte Spalte zeigt die Dropoutraten je LSTM-Zelle an. Der erste Wert stellt den Dropoutwert der ersten LSTM-Zelle, der zweite der zweiten Zelle usw. dar. Diese Werte wurden frei angepasst. Die fünfte Spalte gibt die erreichte Genauigkeit auf die Lerndaten in Prozent, die Sechste die Validierung auf die Testdaten in Prozent, an. Die berechneten Werte wurden auf ganze Zahlen gerundet. Die siebte Spalte gibt an, ob ein Overfitting zu verzeichnen war. Die dick markierten Werte stellen die Bestwerte dar. Wie zu sehen ist, weisen die Optimierer Adagrad und Adadelta keine gute Performance auf, hingegen Adam und RMSprop die beste. Die Aktivierungsfunktion ReLu konnte keine ausreichenden Genauigkeits- und Validierungswerte erzielen. Aus diesem Grund wurden keine Ergebnisse in der

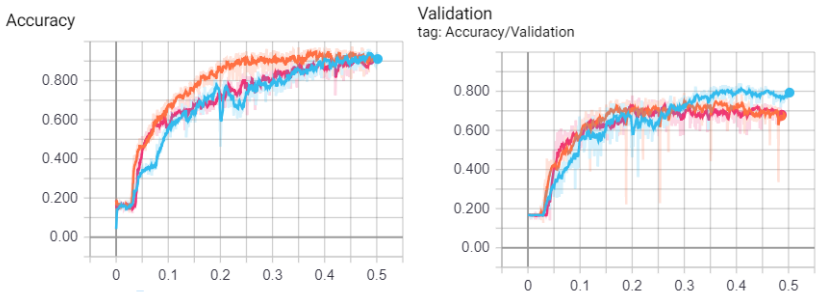


Abbildung 46: Ergebnisse aus der Lern- und Testmenge $(100,\ 25)$
Bei der blauen und orangenen Kurve wurde der Optimierer RMSprop und bei der roten Kurve wurde der Optimierer Adam eingesetzt. Die Daten in der Abbildung weisen eine exponentielle Glättung von $0,8$ auf [5].

Tabelle aufgelistet. Die Optimierer Adagrad und Adadelta konnten in keiner Konfiguration Genauigkeits- und Validierungswerte von über 20% aufweisen und eigneten sich daher nicht zur Klassifizierung der Testmoleküle. Wie zu erkennen ist, weisen die Optimierer Adam und RMSprop die besten Ergebnisse auf, falls eine Lernrate von $0,001$ und die Aktivierungsfunktion Softmax gewählt wurde. Auch bei Lernraten von $0,05$, $0,0005$ und $0,0001$ konnten hierbei gute Ergebnisse erzielt werden. Die Tabelle zeigt ein Auszug der erzielten Ergebnisse.

Die Ergebnisse für das Datenpaar $(100,\ 25)$ sind in Abbildung 46 dargestellt. Bei der blauen Kurve wurde der Optimierer RMSprop und die Aktivierungsfunktion Softmax genutzt. Die erste LSTM-Zelle wies eine Dropoutrate von $0,3$, die zweite eine von $0,4$ und die dritte eine von 0 auf. Die Lernrate betrug $0,001$. Dieses Netzwerk erreichte eine Genauigkeit von 90%. Die Validierung der Daten erfolgte zu 80%. Bei der roten Kurve wurde der Optimierer Adam und die Aktivierungsfunktion Softmax genutzt. Die erste LSTM-Zelle wies eine Dropoutrate von $0,2$, die zweite eine von $0,3$ und die dritte eine von $0,7$ auf. Die Lernrate betrug in diesem Fall ebenfalls $0,001$ und erreichte eine Genauigkeit von 90%. Die Validierung der Daten erfolgte zu 70%. Bei der orangenen Kurve wurde auch der Optimierer RMSprop und die Aktivierungsfunktion Softmax genutzt. Die erste LSTM-Zelle wies eine Dropoutrate von $0,3$, die zweite eine von $0,4$ und die dritte eine von $0,8$ auf. Die Lernrate betrug $0,001$. Dieses Netzwerk erreichte ebenfalls eine Genauigkeit von 90%. Die Validierung der Daten erfolgte lediglich zu 70%. Die Daten in der Abbildung weisen eine exponentielle Glättung von $0,8$ auf [5].

Aufgrund der hohen Anzahl an möglichen Konfigurationen (Wahl der Hyperparameter eines RNN), wurden bei den Datenpaaren $(8.000,\ 2.000)$ und $(500.000,\ 100.000)$ für Lern- und Testdaten nur

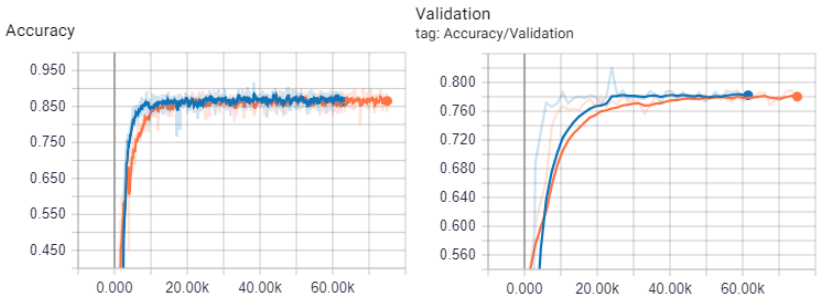


Abbildung 47: Ergebnisse aus der Lern- und Testmenge (8.000, 2.000) Bei der orangenen Kurve wurde der Optimierer RMSprop und bei der blauen Kurve wurde der Optimierer Adam und die Aktivierungsfunktion Softmax genutzt. Die Daten in der Abbildung wiesen eine exponentielle Glättung von 0,8 auf [5].

die Hyperparameter gewählt, die auf den kleineren Datensätzen die besten Ergebnisse erzielen konnten. Die Hyperparameter, die ausgewählt wurden, sind die markierten Werte aus Tabelle 14.

Wie in Abbildung 47 zu erkennen ist, weisen beide Kurven im Datensatz (8.000, 2.000) fast identische Ergebnisse Auf. Bei der orangenen Kurve wurde der Optimierer RMSprop und die Aktivierungsfunktion Softmax genutzt. Die erste LSTM-Zelle wies eine Dropoutrate von 0,3, die zweite eine von 0,4 und die dritte eine von 0,8 auf. Die Lernrate betrug 0,001. Dieses Netzwerk erreichte eine Genauigkeit von 85%. Die Validierung der Daten erfolgte zu 78%. Bei der blauen Kurve wurde der Optimierer Adam und die Aktivierungsfunktion Softmax genutzt. Die erste LSTM-Zelle wies eine Dropoutrate von 0,2, die zweite eine von 0,3 und die dritte eine von 0,7 auf. Die Lernrate betrug in diesem Fall ebenso 0,001 und erreichte eine Genauigkeit von 85%. Sowohl die Genauigkeits- als auch die Validierungskurven konvergierten bei beiden Optimierern bereits sehr schnell in den maximalen Wert und stagnierten dann. Beide Konfigurationen wiesen einen Genauigkeits- und Validierungswert von 85% und 78% auf. Die Validierung der Daten erfolgte zu 78%. Die Daten in der Abbildung wiesen eine exponentielle Glättung von 0,8 auf [5].

In Abbildung 48 wiesen beide Netze im Datensatz (500.000, 100.000) ebenfalls fast identische Ergebnisse auf. Die Hyperparameter, die für beide Instanzen ausgewählt wurden, sind abermals die markierten Werte aus Tabelle 14. Die rote Kurve stellt das Netz mit dem Optimierer Adam dar. Dieses Netz erreichte eine Genauigkeit von 90% und die Validierung der Daten erfolgte zu 93%. Im Netz mit der blauen Kurve wurde der Optimierer RMSprop

Tabelle 14: Ergebnisse RNN – (100, 25)
Die Tabelle zeigt einen Auszug der erzielten Ergebnisse der verschiedenen Optimierer und Aktivierungsfunktionen.

Optimierer	**Aktivierungsfunktion**	**Lernrate (in %)**	**Dropoutraten**	**Genauigkeit (in %)**	**Validierung (in %)**	**Overfitting**
RMSprop	Softmax	0,001	0,1;0,3;0,9	90	70	ja
RMSprop	Softmax	0,001	0,1;0,4;0,9	80	70	ja
RMSprop	Softmax	0,001	0,2;0,3;0,7	90	80	ja
RMSprop	Softmax	0,001	0,2;0,4;0,7	90	70	ja
RMSprop	Softmax	0,001	0,2;0,4;0,9	70	60	nein
RMSprop	**Softmax**	0,001	0,3;0,4;0,8	90	70	**nein**
RMSprop	Softmax	0,0001	0,3;0,4;0,8	80	80	nein
RMSprop	Softmax	0,01	0,3;0,4;0,9	20	20	nein
Adam	ReLu	0,001	0,2;0,3;0,69	20	20	nein
Adam	**Softmax**	0,001	0,2;0,3;0,69	90	70	**nein**
Adam	Softmax	0,001	0,1;0,3;0,9	35	35	ja
Adam	Softmax	0,001	0,1;0,4;0,9	70	60	nein
Adam	Softmax	0,0005	0,2;0,3;0,7	60	60	nein
Adam	Softmax	0,005	0,2;0,3;0,69	16	17	ja
Adam	Softmax	0,05	0,2;0,3;0,7	60	65	ja
Adam	Softmax	0,001	0,2;0,4;0,7	90	65	nein
Adam	Softmax	0,001	0,2;0,4;0,9	45	45	nein
Adam	Softmax	0,001	0,3;0,4;0,8	80	60	nein
Adam	Softmax	0,001	0,3;0,4;0,95	80	55	nein
Adagrad	ReLu	0,001	0,5;0,5;0,5	20	20	nein
Adagrad	Softmax	0,001	0,2;0,3;0,69	17	17	nein
Adadelta	ReLu	0,001	0,5;0,5;0,5	20	20	nein
Adadelta	Softmax	0,001	0,1;0,3;0,9	20	20	nein
Adadelta	Softmax	0,001	0,2;0,7;0,9	20	20	nein

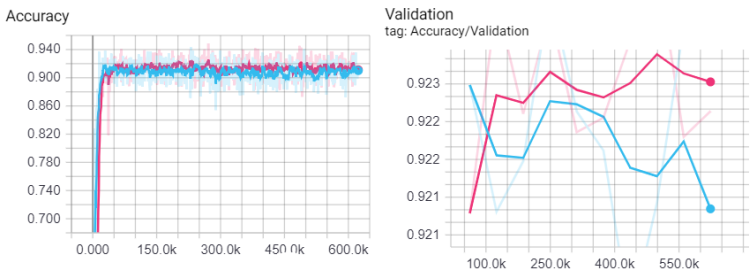


Abbildung 48: Ergebnisse aus der Lern- und Testmenge (500.000, 100.000) Die Hyperparameter, die ausgewählt wurden, sind die markierten Werte aus Tabelle 14. Die rote Kurve stellt das Netz mit dem Optimierer Adam dar. Im Netz mit der blauen Kurve wurde der Optimierer RMSprop eingesetzt. Die Daten in der Abbildung wiesen eine exponentielle Glättung von 0,8 auf [5].

eingesetzt. Der Genauigkeits- und Validierungswert erreichte hier einen Prozentsatz von 90% und 91%. Auch hier ist leicht zu erkennen, dass beide Optimierer ähnliche Ergebnisse erzeugen. Sowohl die Genauigkeits- als auch die Validierungskurven konvergierten auch hier sehr schnell in den maximalen Wert und stagnierten dann. In diesem Fall konvergierte der RMSprop-Optimierer schneller als der Adam-Optimierer. Die Daten in der Abbildung wiesen auch hier eine exponentielle Glättung von 0,8 auf [5].

9.1.4 *Vergleich Graph-Grammatik-Ansatz – Rekurrente Neuronale Netzwerke*

Um die RNN-Ansätze mit denen aus Kapitel 8 bei der Klassifikation von Molekülen vergleichen zu können, wurden die jeweils gleichen Testmengen mit 5.000 und 160 Molekülen genutzt, die auch in Kapitel 8 zur Erzeugung der Graph-Grammatik-Regeln in den Tabellen 10 und 8 zur Substruktursuche eingesetzt wurden. Die Daten wurden in diesem Fall aber nicht wie in Abschnitt 2.5 in eine 80% Lern- und 20% Testdatenmenge aufgeteilt. Die Lernmenge besteht aus 120 Molekülen (20 für jede der sechs Gruppen). Die Lern- und Testdaten setzten sich wie folgt zusammen. Die Lerndaten bestanden aus genau 20 Molekülen pro zu klassifizierender Testgruppe. Es wurden sechs Klassen zur Klassifizierung genutzt, somit wies die Lernmenge insgesamt 120 Moleküle auf. Die Ansätze aus den Abschnitten 8.2 und 8.3 enthielten zum Erzeugen der Graph-Grammatik-Regeln ebenfalls jeweils 20 Moleküle pro Klasse, somit ist ein fairer Vergleich zwischen diesen Ansätzen möglich. Die Testmengen bestehen aus 160 und 5.000 Molekülen, die wie folgt aufgeteilt sind: Die Testmenge mit 160 Molekülen besteht aus jeweils 20 Molekülen pro zu klassifizierender Gruppe plus 40, die keiner dieser angehören. Sie dienen lediglich als zusätzliche

Testelemente. Die zweite Testmenge bestand aus jeweils 750 Moleküle für jede der sechs Gruppen. Hier wurden zusätzlich 500 zufällige Moleküle hinzugefügt. Es sei noch einmal angemerkt, dass die Schnittmenge der Lern- und Testdaten leer ist. Zur Berechnung der Ergebnisse wurden abermals die markierten Hyperparameter aus Tabelle 14 verwendet. Die RMSprop- und Adam-Instanzen mit (120, 160) Lern- und Testdaten wurden 1.000 und 10.000 Epochen, die mit (120, 5.000) Molekülen wurden 1.000 Epochen trainiert.

Die erste Spalte der Ergebnistabellen gibt die zu klassifizierende Gruppe, die zweite und dritte den True-Positive-Wert des Zuordnungswertes des Graph-Grammatik-Ansatzes (TP-GG) und des RNN-Ansatzes (TP-TF), an. Die vierte und fünfte Spalte stellen den jeweiligen True-Negative-Wert dar (TN-GG und TN-TF). In Spalte sechs (FN-GG) und sieben (FN-TF) sind die False-Negative-Werte der beiden Ansätze festgehalten. Spalte neun (FP-GG) und zehn (FP-TF) zeigen die jeweiligen False-Positive-Einträge. Alle Einträge stellen Prozentwerte dar. Die Prozentuale Aufteilung der Positive- und Negative-Werte erfolgte wie in Abschnitt 8.6 beschrieben.

Tabelle 15 stellt die Ergebnisse des Graph-Grammatik-Ansatzes aus Tabelle 8 mit denen des RNN-Ansatzes gegenüber. Hierbei wurde der Optimierer RMS-Prop eingesetzt. Die gewählten Hyperparameter sind die markierten Werte in Tabelle 14 für den RMS-Prop Optimierer. Das RNN Netzwerk wurde hierbei 1.000 Epochen trainiert. Die Lernmenge in beiden Ansätzen betrug jeweils 20 Moleküle pro zu klassifizierender Gruppe. Die Testdaten enthielten jeweils 160 Moleküle. Der Graph-Grammatik-Ansatz erreichte auf allen Testdaten eine Genauigkeit von 100% bei den True-Positive-Werten und nur in zwei Fällen einen True-Negative-Wert, der unter 100% lag. Der RNN-Ansatz konnte bei den Isocyanaten einen True-Positive-Wert von über 90% erreichen. Bei der Klassifizierung der Hydrazone wurde ein Werte von 20% erreicht. Der True-Negative-Wert lag beim RNN-Ansatz bei den Isocyanaten leicht über dem des Graph-Grammatik-Ansatzes, bei ca. 89%.

In Tabelle 16 wurden die gleichen Datensätze wie in Tabelle 15 genutzt. Lediglich der Optimierer und die Hyperparameter wurden durch die Adam-Werte aus Tabelle 14 ersetzt. Das Netzwerk wurde auch hier 1.000 Epochen trainiert. Durch den Wechsel des Optimierers und der dazugehörigen Hyperparameter konnten bei den True-Positive-Werten oft bessere Ergebnisse generiert werden als in Tabelle 15. Lediglich der TP-Wert für Cyanate fiel von ca. 65% auf 45% und der der Isothiocyanate von ca. 85% auf 80%. Kein True-Positive-Wert lag hierbei unter 35%. Der höchste True-Positive-Wert lag bei der Klassifizierung der Isocyanate mit über 95%. Die True-Negative-Werte lagen hier durchweg bei über 79%.

Tabelle 15: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (RNN, RMSProp-Optimierer, 160 Moleküle, 1.000 Epochen)

Gruppe	TP-GG	TP-TF	TN-GG	TN-TF	FN-GG	FN-TF	FP-GG	FP-TF
Hydrazone	100,00	20,14	97,86	91,28	0,00	80,46	2,14	8,63
Sulfate	100,00	50,32	100,00	92,72	0,00	49,68	0,00	7,28
Cyanate	100,00	65,36	100,00	86,25	0,00	35,64	0,00	13,65
Isocyanate	100,00	90,40	87,14	89,13	0,00	9,60	12,86	10,78
Thiocyanate	100,00	75,44	100,00	92,00	0,00	24,56	0,00	7,91
Isothiocyanate	100,00	85,52	100,00	89,13	0,00	14,48	0,00	10,78

Tabelle 16: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (RNN, Adam-Optimierer, 160 Moleküle, 1.000 Epochen)

Gruppe	TP-GG	TP-TF	TN-GG	TN-TF	FN-GG	FN-TF	FP-GG	FP-TF
Hydrazone	100,00	35,20	97,86	94,87	0,00	64,80	2,14	5,13
Sulfate	100,00	65,64	100,00	79,78	0,00	34,36	0,00	20,22
Cyanate	100,00	45,28	100,00	90,56	0,00	54,72	0,00	9,44
Isocyanate	100,00	95,60	87,14	91,28	0,00	4,40	12,86	8,72
Thiocyanate	100,00	75,44	100,00	89,12	0,00	24,56	0,00	10,88
Isothiocyanate	100,00	80,48	100,00	96,41	0,00	19,52	0,00	3,59

In Tabelle 17 wurden die gleichen Datensätze und die gleichen Hyperparameter wie in Tabelle 15 genutzt. Die Datensätze wurden hierbei 10.000 Epochen lang trainiert. Durch die Erhöhung der Trainingsdauer konnte die Testgruppe der Isocyanate zu 100% richtig zugeordnet (TP-FT-Wert) werden. Auch betrug der schlechteste Wert nur noch 40% bei der Hydrazonegruppe. Die sonstigen Werte lagen alle bei über 75%. Der True-Negative-Wert lag auch, bis auf einen in der Isocyanatgruppe, bei über 90%.

Tabelle 17: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (RNN, RMSProp-Optimierer, 160 Moleküle, 10.000 Epochen)

Gruppe	TP-GG	TP-TF	TN-GG	TN-TF	FN-GG	FN-TF	FP-GG	FP-TF
Hydrazone	100,00	15,04	97,86	99,29	0,00	84,96	2,14	0,71
Sulfate	100,00	85,52	100,00	88,41	0,00	14,48	0,00	11,59
Cyanate	100,00	75,44	100,00	89,85	0,00	24,56	0,00	10,15
Isocyanate	100,00	100,00	87,14	89,85	0,00	0,00	12,86	10,06
Thiocyanate	100,00	75,44	100,00	80,50	0,00	24,56	0,00	19,50
Isothiocyanate	100,00	80,48	100,00	99,19	0,00	19,52	0,00	0,81

Tabelle 18: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (RNN, Adam-Optimierer, 160 Moleküle, 10.000 Epochen)

Gruppe	TP-GG	TP-TF	TN-GG	TN-TF	FN-GG	FN-TF	FP-GG	FP-TF
Hydrazone	100,00	40,24	97,86	90,56	0,00	59,76	2,14	9,44
Sulfate	100,00	80,48	100,00	92,72	0,00	19,56	0,00	7,28
Cyanate	100,00	75,44	100,00	93,44	0,00	24,56	0,00	6,56
Isocyanate	100,00	100,00	87,00	76,90	0,00	0,00	12,86	23,00
Thiocyanate	100,00	75,44	100,00	97,03	0,00	24,56	0,00	2,88
Isothiocyanate	100,00	85,52	100,00	99,91	0,00	14,48	0,00	0,00

Tabelle 19: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (RNN, RMSProp-Optimierer, 5.000 Moleküle, 1.000 Epochen)

Gruppe	TP-GG	TP-TF	TN-GG	TN-TF	FN-GG	FN-TF	FP-GG	FP-TF
Hydrazone	99,86	47,34	94,02	94,56	0,14	52,66	5,97	5,44
Sulfate	99,87	18,78	99,95	94,60	0,13	81,22	0,05	5,40
Cyanate	99,88	59,76	99,88	83,66	0,16	40,24	0,12	16,34
Isocyanate	99,75	81,78	97,05	90,07	0,25	18,24	2,95	9,93
Thiocyanate	99,73	94,98	99,87	96,74	0,27	5,02	0,13	3,26
Isothiocyanate	99,48	59,22	99,15	92,74	0,54	40,78	0,85	7,26

In Tabelle 18 wurden in diesem Fall auch die gleichen Datensätze und Hyperparameter wie in Tabelle 16 genutzt. Die Datensätze wurden auch hier 10.000 Epochen lang trainiert. Durch die Erhöhung der Trainingsdauer konnte die Testgruppe der Isocyanate zu 100% richtig zugeordnet werden. Die Werte lagen alle bei über 75%, außer bei der Hydrazonegruppe. Der True-Negative-Wert lag auch, ausgenommen die Isocyanatgruppe, bei über 90% (siehe TP-TF-Werte) .

In den Tabellen 19 und 20 werden die erzeugten Ergebnisse aus Tabelle 10 abermals mit den RNN-Ansätzen verglichen. Hierbei wurde bisher der Optimierer RMS-Prop in Tabelle 19 und der Adam Optimierer in Tabelle 20 eingesetzt. Die gewählten Hyperparameter sind die gleichen wie in Tabelle 15 und 16. Die Datenaufteilung ist identisch mit der aus Tabelle 10. Die RNN-Netzwerke wurden hierbei 1.000 Epochen trainiert. Wie man den Tabellen entnehmen kann, verschlechterten sich fast alle Werte in Bezug auf die Tabellen 17 und 18. Die Testmengen wurden hier von 160 auf 5.000 Moleküle erhöht, die Lernmengen blieben aber konstant bei 120 Molekülen.

Tabelle 20: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (RNN, Adam-Optimierer, 5.000 Moleküle, 1.000 Epochen)

Gruppe	TP-GG	TP-TF	TN-GG	TN-TF	FN-GG	FN-TF	FP-GG	FP-TF
Hydrazone	99,86	73,20	94,02	90,92	0,14	26,80	5,97	9,07
Sulfate	99,87	18,94	99,95	97,09	0,13	81,06	0,05	2,90
Cyanate	99,88	69,04	99,91	82,92	0,12	30,96	0,09	17,08
Isocyanate	99,75	64,78	97,05	84,95	0,25	35,22	2,95	15,05
Thiocyanate	99,73	79,50	99,19	97,64	0,27	20,5	0,80	2,35
Isothiocyanate	99,46	23,34	99,14	92,21	0,54	76,66	0,85	7,79

Tabelle 21: Ergebnisse – Random Forest Verfahren, 160 Moleküle, 10 Bäume und eine Baumtiefe von 10

Gruppe	$\varnothing_{TP}$	σ_{TP}	$\varnothing_{TN}$	σ_{TN}	$\varnothing_{FN}$	σ_{FN}	$\varnothing_{FP}$	σ_{FP}
Ketone	35,50	0,93	84,25	2,29	64,50	2,29	15,75	0,93
Konj.-Ester	40,85	1,32	87,55	2,05	59,15	2,05	12,45	1,32
Carbonsäureamide	30,61	0,99	83,38	2,49	69,39	2,49	16,62	0,99
C-Nitro	39,60	0,98	92,14	1,70	60,40	1,70	7,86	0,98
Oxime	74,29	1,13	91,74	1,84	25,71	1,84	8,26	1,13
Nitrile	46,65	1,87	94,48	1,11	53,35	1,11	5,52	1,87

9.2 KLASSIFIZIERUNG MITTELS RANDOM FOREST

Zur Klassifizierung wird in diesem Abschnitt der Random Forest Ansatz aus Abschnitt 2.5.3 genutzt. Hierbei werden die gleichen Datensätze mit der gleichen Aufteilung wie in Abschnitt 9.1.2 eingesetzt. Aus den Datensätzen wurden die Mittelwerte ($\varnothing_{TP}, \varnothing_{TN}, \varnothing_{FN}, \varnothing_{FP}$) und die jeweiligen Standardabweichungen (σ_{TP}, σ_{TN}, σ_{FN}, σ_{FP}) in Prozent aus 1.000 Experimenten für jede Klasse berechnet und in einer Tabelle festgehalten. Da es sich, wie in Abschnitt 2.5.3 beschrieben, um ein statistisches Verfahren handelt, wurde a priori erwartet, dass der Wert der Standardabweichung sehr gering bei der Berechnung der Ergebnisse ausfallen wird. Dieser wurde nur zur Überprüfung der Daten zusätzlich berechnet.

9.2.1 *Ergebnisse*

Bei der Optimierung müssen im Random Forest Verfahren ebenfalls Hyperparameter definiert werden, wie z. B. die Anzahl an genutzten Bäumen, die maximale Baumtiefe und die maximale Anzahl an Blattknoten. Die Hyperparameter, die zur Optimierung gewählt wurden, sind die maximale Baumtiefe und die Anzahl an genutzten Bäumen.

Tabelle 22: Ergebnisse – Random Forest Verfahren, 160 Moleküle, 100 Bäume und eine Baumtiefe von 50

Gruppe	$\varnothing_{TP}$	σ_{TP}	$\varnothing_{TN}$	σ_{TN}	$\varnothing_{FN}$	σ_{FN}	$\varnothing_{FP}$	σ_{FP}
Ketone	38,75	0,60	88,91	1,12	61,25	1,12	11,09	0,60
Konj,-Ester	49,05	0,54	90,73	0,90	50,95	0,90	9,27	0,54
Carbonsäureamide	36,14	0,65	86,24	1,40	63,86	1,40	13,76	0,65
C-Nitro	53,12	0,60	94,33	0,79	46,88	0,79	5,67	0,60
Oxime	84,99	0,32	91,55	0,97	15,01	0,97	8,45	0,32
Nitrile	63,32	0,72	93,35	0,69	36,68	0,69	6,65	0,72

Zur Klassifizierung wurden als Hyperparameter die Anzahl der Bäume und die Baumtiefe mit den Werten $(10, 10)$ und $(100, 50)$ verwendet. Die prozentuale Aufteilung der Positive- und Negative-Werte erfolgte wie in Abschnitt 8.6 beschrieben. Bei der Klassifizierung mit dem Random Forest Verfahren konnten bereits bei einer Nutzung von 10 Bäumen und einer maximalen Baumtiefe von 10 gemittelte True-Positive-Werte von über 84% in der Gruppe Oxime erreicht werden. Der True-Negative-Anteil wies immer Werte von über 85% auf. Die Standardabweichung σ besaß hier einen Höchstwert von $2,50$. Die Testmenge bestand in diesem Fall aus jeweils 25 Elementen pro zu klassifizierender Gruppe und die Lernmenge aus jeweils 100 Molekülen pro Gruppe. Eine Übersicht der Ergebnisse ist in Tabelle 21 festgehalten.

In Tabelle 22 wurden die gleichen Daten mit der gleichen Aufteilung genutzt. Lediglich der Wert für die Anzahl an Bäumen und die maximale Baumtiefe wurde auf 100 und 50 erhöht. Dabei konnten alle True-Positive-Werte in der Tabelle verbessert werden. Die höchste Verbesserung beim TP-Wert konnte die Gruppe Nitrile erzielen ($16, 67\%$).

In Tabelle 23 wurden die Daten wie folgt aufgeteilt. Die Lernmenge bestanden hier aus 8.000 Molekülen und die Testmenge aus 2.000 pro Gruppe. Die Hyperparameter für die Anzahl an Bäumen und Baumtiefe betrugen hier jeweils 10. Durch die Erhöhung der Datenmenge konnten einige True-Positive-Werte aus Tabelle 22 erheblich verbessert werden. Der Nitrile-Wert konnte von ca. 63% auf über 81% gesteigert werden. Bei der Gruppe für konjugierte Ester verschlechterte sich der True-Positive-Anteil von ca. 49% auf 32%.

In Tabelle 24 betrug der Hyperparameter für die Anzahl an genutzten Bäumen und der Baumtiefe wieder 100 und 50. Durch die Erhöhung der Hyperparameter konnten, im Vergleich zur Tabelle 23, alle True-Positive auf bis zu 81% und True-Negative-Werte auf bis zu 98% gesteigert werden.

Tabelle 23: Ergebnisse – Random Forest Verfahren, 8.000 Moleküle, 10 Bäume und eine Baumtiefe von 10

Gruppe	$\varnothing_{TP}$	σ_{TP}	$\varnothing_{TN}$	σ_{TN}	$\varnothing_{FN}$	σ_{FN}	$\varnothing_{FP}$	σ_{FP}
Ketone	41,02	0,25	86,18	0,48	58,98	0,48	13,82	0,25
Konj,-Ester	32,37	0,21	94,63	0,19	67,63	0,19	5,37	0,21
Carbonsäureamide	59,52	0,15	87,12	0,60	40,48	0,60	12,88	0,15
C-Nitro	66,16	0,24	91,87	0,32	33,84	0,32	8,13	0,24
Oxime	80,45	0,12	98,38	0,17	19,55	0,17	1,62	0,12
Nitrile	81,06	0,12	94,35	0,40	18,94	0,40	5,65	0,12

Tabelle 24: Ergebnisse – Random Forest Verfahren, 8.000 Moleküle, 100 Bäume und eine Baumtiefe von 50

Gruppe	$\varnothing_{TP}$	σ_{TP}	$\varnothing_{TN}$	σ_{TN}	$\varnothing_{FN}$	σ_{FN}	$\varnothing_{FP}$	σ_{FP}
Ketone	52,40	0,08	87,55	0,13	47,6	0,13	12,45	0,08
Konj,-Ester	42,89	0,07	95,87	0,06	57,11	0,06	4,13	0,07
Carbonsäureamide	74,90	0,04	90,23	0,17	25,10	0,17	9,77	0,04
C-Nitro	76,00	0,06	95,10	0,06	24,00	0,06	4,90	0,06
Oxime	83,17	0,05	98,59	0,05	16,83	0,05	1,41	0,05
Nitrile	83,00	0,03	94,93	0,07	17,00	0,07	5,07	0,03

In den Tabellen 25 und 26 wurden statt der zuvor sechs genutzten Klassen aus Gründen der Datenmenge nur vier untersucht. Die Klassen der Oxime und konjugierten Ester wurden hierbei, wie in Abschnitt 9.1.2, entfernt. Die Datensätze enthielten in der Lernmenge jeweils 500.000 und in der Testmenge 100.000 Elemente pro Klasse. Hier konnte das Random Forest Verfahren, bei der Nutzung von 10 Bäumen und einer maximalen Baumtiefe von 10, in allen Gruppen nur einen maximalen TP-Wert von 55% erreicht werden. Lediglich der Wert für die Gruppe der Nitrilen betrug 73%. Der True-Positive-Anteil konnte in zwei Gruppen auf über 90% in den Gruppen C-Nitro und Nitrile gesteigert werden.

Der Random Forest Ansatz konnte ebenso wie der RNN-Ansatz erfolgreich zur Klassifikation von Molekülen eingesetzt werden. Dabei wurden True-Positive-Werte von bis zu 85% erreicht. Das Random Forest Verfahren benötigt im Gegensatz zum RNN-Ansatz nur einen Bruchteil an CPU-Zeit, um bereits gute Ergebnisse zu erzeugen. Das gilt auch bei kleinen Datensätzen. Im Random Forest Verfahren wurde beobachtet, dass durch die Erhöhung der Hyperparameter die Zuordnungswerte nur bis zu einem bestimmten Wert gesteigert werden konnten. Sie verblieben dann in diesem Wert. Der Random Forest

Tabelle 25: Ergebnisse – Random Forest Verfahren, 500.000 Moleküle, 10 Bäume und eine Baumtiefe von 10

Gruppe	$\varnothing_{TP}$	σ_{TP}	$\varnothing_{TN}$	σ_{TN}	$\varnothing_{FN}$	σ_{FN}	$\varnothing_{FP}$	σ_{FP}
Ketone	46,89	0,31	79,47	0,42	53,11	0,42	20,53	0,31
Carbonsäureamide	59,65	0,38	77,86	0,70	40,35	0,70	22,14	0,38
C-Nitro	54,06	0,44	92,75	0,26	45,94	0,26	7,25	0,44
Nitrile	73,09	0,30	94,50	0,58	26,91	0,58	5,50	0,30

Tabelle 26: Ergebnisse – Random Forest Verfahren, 500.000 Moleküle, 100 Bäume und eine Baumtiefe von 50

Gruppe	$\varnothing_{TP}$	σ_{TP}	$\varnothing_{TN}$	σ_{TN}	$\varnothing_{FN}$	σ_{FN}	$\varnothing_{FP}$	σ_{FP}
Ketone	70,97	0,03	85,02	0,04	29,03	0,04	14,98	0,03
Carbonsäureamide	69,64	0,03	88,53	0,04	30,36	0,04	11,47	0,03
C-Nitro	79,08	0,07	95,89	0,03	20,92	0,03	4,11	0,07
Nitrile	75,59	0,02	95,68	0,04	24,41	0,04	4,32	0,02

Ansatz ist ein gutes Verfahren, um in kurzer Zeit gute Ergebnisse zu generieren.

9.2.2 *Vergleich Graph-Grammatik-Ansatz – Random Forest*

Um den Random Forest Ansatz ebenfalls mit den Graph-Grammatik-Ansätzen aus Kapitel 8 bei der Klassifikation von Molekülen vergleichen zu können, wurden die jeweils gleichen Testmengen mit 5.000 und 160 Molekülen genutzt, die auch in Kapitel 8 zur Erzeugung der Graph-Grammatik-Regeln in den Tabellen 8 und 10 bei der Berechnung eingesetzt wurden. Es wurden die gleichen Daten mit der gleichen Aufteilung wie in Abschnitt 9.1.4 zum Vergleich genutzt. Zur Berechnung der Random Forest Ergebnisse wurden die Hyperparameter der genutzten Bäume auf 100 und die der Baumtiefe auf 50 gesetzt.

Die Lernmengen zur Berechnung der Ergebnisse in den Tabellen 27 und 28 wiesen jeweils 20 Moleküle pro Klasse auf. Die Testmengen bestanden jeweils aus insgesamt 160 und 5.000 Molekülen. In den Tabellen 27 und 28 ist zu erkennen, dass der Random Forest Ansatz bei einer Lernmenge von jeweils 20 Molekülen nicht die Ergebnisse des Graph-Grammatik- oder RNN-Ansatzes erreichen konnte. Lediglich

Tabelle 27: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (Random Forest Verfahren, 160 Moleküle, 100 Bäume, Baumtiefe 50)

Gruppe	TP-GG	$\varnothing_{TP}$	σ_{TP}	TN-GG	$\varnothing_{TN}$	σ_{TN}	FN-GG	$\varnothing_{FN}$	σ_{FN}	FP-GG	$\varnothing_{FP}$	σ_{FP}
Hydrazone	100,00	34,71	0,54	97,86	89,28	1,01	0,00	65,29	1,01	2,14	10,72	0,54
Sulfate	100,00	46,17	0,43	100,00	88,67	0,46	0,00	53,83	0,46	0,00	11,33	0,43
Cyanate	100,00	34,41	0,49	100,00	85,83	1,05	0,00	65,59	1,05	0,00	14,17	0,49
Isocyanate	100,00	36,75	0,49	87,14	85,22	1,76	0,00	63,25	1,76	12,86	14,78	0,49
Thiocyanate	100,00	79,11	1,54	100,00	92,52	0,45	0,00	20,89	0,45	0,00	7,48	1,54
Isothiocyanate	100,00	22,08	0,35	100,00	89,15	1,21	0,00	77,92	1,21	0,00	10,85	0,35

Tabelle 28: Vergleich Graph-Grammatik-Ansatz – Maschinelles Lernen (Random Forest Verfahren, 5.000 Moleküle, 100 Bäume, Baumtiefe 50)

Gruppe	TP-GG	$\varnothing_{TP}$	σ_{TP}	TN-GG	$\varnothing_{TN}$	σ_{TN}	FN-GG	$\varnothing_{FN}$	σ_{FN}	FP-GG	$\varnothing_{FP}$	σ_{FP}
Hydrazone	99,86	35,71	0,9	94,02	86,71	2,07	0,14	64,29	2,07	5,97	13,29	0,9
Sulfate	99,87	16,40	0,30	99,95	90,18	1,28	0,13	83,60	1,28	0,05	9,82	0,30
Cyanate	99,88	30,19	0,62	99,88	80,83	2,46	0,16	69,81	2,46	0,12	19,17	0,62
Isocyanate	99,75	33,99	0,79	97,05	76,86	3,18	0,25	66,01	3,18	2,95	23,14	0,79
Thiocyanate	99,73	41,23	1,65	99,87	94,26	1,11	0,27	58,77	1,11	0,13	5,74	1,65
Isothiocyanate	99,48	18,79	0,47	99,15	86,42	2,55	0,54	81,21	2,55	0,85	13,58	0,47

im Fall der Thiocyanate (Tabelle 25) wurde ein Wert von über 79% erreicht.

9.3 GESAMTVERGLEICH GRAPH-GRAMMATIK – MASCHINELLES LERNEN

Das maschinelle Lernen unter Zuhilfenahme von RNNs und des Random Forest Verfahrens konnte erfolgreich zur Klassifizierung von Molekülen eingesetzt werden. Das RNN-Verfahren konnte bereits bei sehr flachen Netzen gute Ergebnisse erzielen. Hierbei konnten die Optimierer Adam und RMSprop bei Lernraten von $0,001$, $0,0001$, $0,0005$ und $0,05$ gute Ergebnisse erzielen. Als Aktivierungsfunktion sollte Softmax eingesetzt werden, bei den übrigen konnten keine Werte von über 20% erreicht werden. Wiesen die Werte der LSTM-Zellen unterschiedliche Dropoutwerte auf, konnten ebenfalls bessere Werte berechnet werden. Diese mussten aber adaptiv an die Daten angepasst werden. Somit konnten Genauigkeits- und Validierungwerte von über 80% erreicht werden. Die Optimierer Adagrad und Adadelta hingegen konnten mit keinen der gewählten Hyperparametern und Aktivierungsfunktionen gute Ergebnisse aufweisen. Beim Vergleich mit den Ansätzen aus den Abschnitten 8.2 und 8.3 wiesen

die RNN-Netze durchweg schlechtere Ergebnisse auf als die Graph-Grammatik-Ansätze. Die RNN-Ansätze konnten die Daten über eine gewisse Anzahl an Epochen analysieren, die Ansätze aus Kapitel 8 erzeugten die Regeln in nur einem Durchlauf.

Im Allgemeinen ist das maschinelle Lernen ein gutes Werkzeug zum Klassifizieren von Molekülen. Bereits mit kleinen Datensätzen können gute Ergebnisse erzielt werden. Bei RNNs sind jedoch viele Konfigurationen nötig, wie das Auswählen des Optimierers, der Aktivierungsfunktion und das Einstellen der Hyperparameter, um diese zu erreichen. Wenn RNN-Netze auf eine Problemstellung angelernt werden, liefern sie gute bis sehr gute Ergebnisse. Das Random Forest Verfahren kann ohne erhebliche Anpassung der Hyperparameter ebenfalls gute Ergebnisse erreichen. Positiv hier ist auch die benötigte CPU-Zeit. Das Random Forest Verfahren konnte ohne große Anpassung der Hyperparameter in einem Bruchteil der RNN-Zeit gute Ergebnisse erzielen. Lediglich die Anzahl der Bäume und der Parameter für die Baumtiefen wurde adaptiv an die Daten angepasst. Bei einer Lernmenge von 25 Molekülen konnte das Random Forest Verfahren die Zuordnungswerte des Graph-Grammatik-Ansatzes nicht erreichen.

Abschließend ist festzuhalten, dass das RNN-Verfahren genauere Zuordnungswerte erzielt, der Random Forest Ansatz Ergebnisse erheblich schneller generiert, bei oft nicht wesentlich schlechteren Ergebnissen. Der Anwender sollte beim Klassifizieren von Molekülen diesen Trade-Off beachten.

10 FAZIT UND AUSBLICK

Das Ziel dieser Arbeit war es, einen Graph-Grammatik-Ansatz zur Substruktursuche oder Klassifikation von Molekülen vorzustellen und zu implementieren. Des Weiteren wurde die Komplexität des präsentierten Algorithmus untersucht. Es wurden unterschiedliche Ansätze zum Erlernen von Graph-Grammatiken vorgestellt. Die präsentierten Ansätze zum Erlernen der Graph-Grammatiken wurden mit den RNN- und dem Random Forest Ansatz aus dem Bereich des maschinellen Lernens miteinander verglichen. Hierbei sollte aufgezeigt werden, welche Ansätze besser zur Klassifikation von Molekülen geeignet sind. Die zu untersuchenden Moleküldatensätze stammten aus der Zinc [76] und PubChem-Datenbank [95].

In dieser Arbeit wurde ein Algorithmus zur molekularen Substruktursuche auf Graphen vorgestellt. Der Algorithmus weist eine polynomielle Laufzeit auf, falls der Grad der Ersetzungsregeln des Graphersetzungssystems und die Größe des Schnittes zwischen jedem Paar an Zusammenhangskomponenten beschränkt sind. Es wurde gezeigt, dass der vorgestellte Ansatz auf Bäumen $\mathcal{W}[1]$-schwer sowie auf Bäumen mit beschränktem Knotengrad und beschränkter Baumweite $\mathcal{NP}$-vollständig ist.

Es wurden unterschiedliche Algorithmen zum Lernen von Graph-Grammatik-Regeln vorgestellt. Die hierbei erzeugten Regeln wurden unter Zuhilfenahme des Algorithmus aus Kapitel 6 zur Substruktursuche oder zur Klassifikation eingesetzt. Die erzeugten Regeln der unterschiedlichen Ansätze haben sich in den Testszenarien als effizient erwiesen und konnten gute bis sehr gute Ergebnisse aufweisen. Bereits mit den erzeugten Regeln des Algorithmus aus Abschnitt 8.2 konnte für die Gruppe der Sulfate ein True-Positive-Wert von 100%, unter Zuhilfenahme des Algorithmus aus Kapitel 6, berechnet werden. Durch den Einsatz des Algorithmus aus 8.2 und die zusätzliche Nutzung des Generalisierungsalgorithmus aus Abschnitt 8.4 verbesserten sich in fast allen Testfällen die Zuordnungswerte. Alle Gruppen wiesen hierbei hohe True-Positive- und True-Negative-Werte bei der Substruktursuche auf (siehe Tabelle 8). Zusätzlich konnte durch den Algorithmus aus Abschnitt 8.4 die Anzahl an genutzten Graph-Grammatik-Regeln erheblich verringert werden. Der False-Negative-Anteil wies in der Kombination der Algorithmen aus 8.2 und 8.4 immer annähernd einen Wert von 0% auf. Bei Vernachlässigung der Knotengrade konnten fast alle Moleküle der richtigen Testgruppe zugeordnet werden. Im Allgemeinen ist festzuhalten, dass das Ver-

nachlässigen der Knotengrade dabei helfen kann, Substrukturen ihrer Gruppe zuzuordnen, dadurch aber viele False-Positive-Fehler zusätzlich generiert werden (siehe Tabelle 9).

Mit dem Algorithmus aus Abschnitt 8.3, der unter Einbeziehung von chemischen Aspekten Graph-Grammatik-Regeln generiert, konnten ebenfalls gute bis sehr gute Werte bei der Substruktursuche erzielt werden. Die Validierungswerte lagen oft bei über 90%.

Abschließend ist festzuhalten, dass die erzeugten Graph-Grammatik-Regeln durch die vorgestellten Ansätze zum Lernen von Graph-Grammatiken erfolgreich zur Substruktursuche eingesetzt werden konnten. Bereits mit einer kleinen Lernmenge wurden Graph-Grammatik-Regeln erzeugt, die nahezu alle Moleküle entweder richtig klassifizieren oder die gesuchten Substrukturen erfolgreich identifizieren konnten. In Zukunft sollten weitere funktionale Gruppen mit den Graph-Grammatik-Ansätzen untersucht werden.

Zukünftig sollten die Ansätze aus Kapitel 8 durch weitere chemische Gegebenheiten erweitert werden, d. h. durch zusätzliche Eigenschaften, die speziell sind oder sehr selten in einem Molekül auftreten. Des Weiteren sollten Algorithmen entwickelt werden, die Graph-Grammatik-Regeln lernen und die Strukturen beliebiger Länge erzeugen können, wie z. B. die von unterschiedlichen sich wiederholenden String-Sequenzen.

Die vorgestellten Ansätze des maschinellen Lernens aus Kapitel 9 konnten, ebenso wie die Graph-Grammatik-Ansätze, erfolgreich zur Klassifikation von Molekülen genutzt werden. Einige Testgruppen im RNN-Verfahren konnten bis zu 100% richtig zugeordnet werden. Die Optimierer Adam und RMSProp berechneten hierbei wesentlich bessere Werte als die anderen hier getesteten. Bei der Wahl der Hyperparameter fiel auf, dass Lernraten von $0,0001$, $0,001$, $0,05$ und $0,005$ erfolgversprechender sind als andere. Das Random Forest Verfahren wies bei einer Baumweite von 50 und einer maximalen Anzahl an 100 Bäumen ebenfalls Werte von über 90% auf.

Um die Ergebnisse der Ansätze des maschinellen Lernens aus Kapitel 9 mit denen aus Kapitel 8 bei der Klassifikation von Molekülen vergleichen zu können, wurden die jeweils gleichen Lern- und Testdaten genutzt, die auch in Kapitel 8 zur Regelerzeugung und Klassifikation angewandt wurden. Alle zu vergleichenden Ansätze bekamen die gleichen Lern- und Testdaten. Beim Vergleich standen nun nur noch 20 Moleküle pro zu klassifizierender chemisch funktionaler Gruppe zur Verfügung. Dabei konnte sowohl der RNN- als auch der Random Forest Ansatz nicht die Ergebnisse der Graph-Grammatik-Ansätze erreichen.

Es wurde gezeigt, dass im Falle von nur wenigen Lerndaten zur Klassifizierung von Molekülen die Ansätze aus Kapitel 8 bessere Zuordnungswerte erzeugen als die Ansätze des maschinellen Lernens aus Kapitel 9. Es sei angemerkt, dass die Ansätze aus Kapitel 8 nur einen Durchlauf bei der Erzeugung der Graph-Grammatik-Regeln benötigen. Die RNN-Ansätze müssen mit bis zu 10.000 Epochen trainiert werden, um annähernd gute Ergebnisse zu erzielen. Die Ansätze aus Kapitel 8 benötigten lediglich 20 Moleküle pro gesuchter funktionaler Gruppe, um fast alle Elemente mit einer hohen Genauigkeit zuzuordnen. Auch sind die Ansätze einfacher zu nutzen, d. h. es müssen nicht unzählige Hyperparameter und Optimierer bei der Klassifizierung getestet werden, bis gute Ergebnisse erzielt werden. Dennoch können mit den Ansätzen des maschinellen Lernens ebenfalls gute bis sehr gute Ergebnisse beim Klassifizieren von Molekülen erzielt werden. Es sind jedoch viele Einstellungen, unter anderem der Hyperparameter, nötig, um diese zu erreichen. Man kann a priori nicht sagen, welche zum Erfolg führen. Wenn ein Modell auf eine Problemstellung angelernt wurde, d. h. die Hyperparameter angepasst wurden, liefern diese Ansätze aber gute bis sehr gute Ergebnisse. Das RNN-Verfahren kann genauere Zuordnungswerte berechnen. Der Random Forest Ansatz generiert Ergebnisse erheblich schneller, bei oft nicht wesentlich schlechteren Endergebnissen. Der Anwender sollte beim Klassifizieren von Molekülen diesen Trade-Off beachten.

Als mögliche Erweiterung in zukünftigen Arbeiten zur Nutzung der Ansätze des maschinellen Lernens zur Klassifikation von Molekülen sollten zusätzliche Hyperparameter definiert und adaptiv auf die Daten angepasst werden. Dies gilt sowohl für das RNN- als auch das Random Forest Modell.

[1] Bayer AG. *Kleine und große Moleküle*. 2019. URL: http://pharma.bayer.com/de/innovation-und-partnerschaften/technologien-trends/kleine-grosse-molekuele/ (besucht am 27. 07. 2019).

[2] LUMITOS AG. *CHEMIE.DE*. 2019. URL: https://www.chemie.de (besucht am 17. 06. 2019).

[3] Karl A Abrahamson, Rodney G Downey und Michael Ralph Fellows. "Fixed-parameter tractability and completeness IV: On completeness for W [P] and PSPACE analogues". In: *Annals of pure and applied logic* 73.3 (1995), S. 235–276.

[4] Ernst Althaus, Andreas Hildebrandt und Domenico Mosca. "Graph Rewriting Based Search for Molecular Structures: Definitions, Algorithms, Hardness". In: *Federation of International Conferences on Software Technologies: Applications and Foundations*. Springer. 2017, S. 43–59.

[5] The TensorFlow Authors. *Tensorflow documentation*. 2018. URL: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/overfit_and_underfit.ipynb (besucht am 24. 01. 2019).

[6] Dzmitry Bahdanau, Kyunghyun Cho und Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[7] Tadas Baltrušaitis, Chaitanya Ahuja und Louis-Philippe Morency. "Multimodal machine learning: A survey and taxonomy". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.2 (2019), S. 423–443.

[8] Marian Stewart Bartlett, Gwen Littlewort, Mark Frank, Claudia Lainscsek, Ian Fasel und Javier Movellan. "Recognizing facial expression: machine learning and application to spontaneous behavior". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Bd. 2. IEEE. 2005, S. 568–573.

[9] Allen H Barton. "The concept of property-space in social research". In: *The language of social research* (1955), S. 40–53.

[10] Michel Bauderon und Bruno Courcelle. "Graph expressions and graph rewritings". In: *Mathematical Systems Theory* 20.1 (1987), S. 83–127.

[11] Dan Becker. *Rectified Linear Units (ReLU) in Deep Learning*. 2018. URL: https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning (besucht am 23. 01. 2019).

[12] Norman Biggs, Norman Linstead Biggs und Biggs Norman. *Algebraic graph theory*. Bd. 67. Cambridge university press, 1993.

[13] Universität Hamburg Center for Bioinformatics. *SMARTSviewer*. 2019. URL: https://smarts.plus/ (besucht am 10. 04. 2019).

[14] Léon Bottou, Frank E Curtis und Jorge Nocedal. "Optimization methods for large-scale machine learning". In: *Siam Review* 60.2 (2018), S. 223–311.

[15] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), S. 123–140.

[16] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), S. 5–32.

[17] Leo Breiman und Ross Ihaka. *Nonlinear discriminant analysis via scaling and ACE*. Department of Statistics, University of California, 1984.

[18] Leo Breiman, Jerome Friedman, Richard Olshen und Charles Stone. "Classification and regression trees." In: *Group* 37.15 (1984), S. 237–251.

[19] Alain Bretto. "Hypergraph theory". In: *An introduction. Mathematical Engineering. Cham: Springer* (2013).

[20] Annalisa Buffa, Yvon Maday, Anthony T Patera, Christophe Prud'homme und Gabriel Turinici. "A priori convergence of the greedy algorithm for the parametrized reduced basis method". In: *ESAIM: Mathematical modelling and numerical analysis* 46.3 (2012), S. 595–603.

[21] Robert Burbidge, Matthew Trotter, B Buxton und Sl Holden. "Drug design by machine learning: support vector machines for pharmaceutical data analysis". In: *Computers & chemistry* 26.1 (2001), S. 5–14.

[22] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk und Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[23] Pedro Cintas. "Ursprünge und Entwicklung der Begriffe Chiralität und Händigkeit in der chemischen Sprache". In: *Angewandte Chemie* 119.22 (2007), S. 4090–4099.

[24] Volker Claus, Hartmut Ehrig und Grzegorz Rozenberg. *Graph-grammars and their application to computer science and biology: international workshop, Bad Honnef, October 30-November 3, 1978*. Bd. 1. Springer Science & Business Media, 1979.

[25] A Cochocki und Rolf Unbehauen. *Neural networks for optimization and signal processing*. John Wiley & Sons, Inc., 1993.

[26] Luigi P Cordella, Pasquale Foggia, Carlo Sansone und Mario Vento. "A (sub) graph isomorphism algorithm for matching large graphs". In: *IEEE transactions on pattern analysis and machine intelligence* 26.10 (2004), S. 1367–1372.

[27] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest und Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[28] W. E. Cossum, M. L. Krakiwsky und Michael F. Lynch. "Advances in Automatic Chemical Substructure Searching Techniques." In: *Journal of Chemical Documentation* 5.1 (1965), S. 33–35.

[29] Bruno Courcelle. "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs". In: *Information and computation* 85.1 (1990), S. 12–75.

[30] Bruno Courcelle. "Context-free graph grammars: Separating vertex replacement from hyperedge replacement". In: *International Symposium on Fundamentals of Computation Theory*. Springer. 1993, S. 181–193.

[31] Joseph A Cruz und David S Wishart. "Applications of machine learning in cancer prediction and prognosis". In: *Cancer informatics* 2 (2006), S. 117693510600200030.

[32] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk und Saket Saurabh. *Parameterized algorithms*. Bd. 4. 8. Springer, 2015.

[33] Ivan Nunes Da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni und Silas Franco dos Reis Alves. "Artificial neural networks". In: *Cham: Springer International Publishing* (2017).

[34] Aymeric Damien u. a. *TFLearn*. https://github.com/tflearn/tflearn. 2016. (Besucht am 24. 01. 2019).

[35] AK Dehof, HP Lenhof und A Hildebrandt. "Predicting protein NMR chemical shifts in the presence of ligands and ions using force field-based features". In: *Proceedings of the German Conference on Bioinformatics 2011*. Weihenstephan. 2011.

[36] Anna Katharina Dehof, Alexander Rurainski, Quang Bao Anh Bui, Sebastian Böcker, Hans-Peter Lenhof und Andreas Hildebrandt. "Automated bond order assignment as an optimization problem". In: *Bioinformatics* 27.5 (2011), S. 619–625.

[37] Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi und Dimitrios M Thilikos. "Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs". In: *Journal of the ACM (JACM)* 52.6 (2005), S. 866–893.

[38] Reinhard Diestel. *Graphentheorie (3. Aufl.).* 2006.

[39] Matthias Dietzen, Elena Zotenko, Andreas Hildebrandt und Thomas Lengauer. "Correction to on the applicability of elastic network normal modes in small-molecule docking". In: *Journal of chemical information and modeling* 54.12 (2014), S. 3453–3453.

[40] Rodney G Downey und Michael Ralph Fellows. "Fixed-parameter tractability and completeness". In: *Congressus Numerantium* (1992), S. 161–161.

[41] Rodney G Downey und Michael Ralph Fellows. "Fixed-parameter tractability and completeness I: Basic results". In: *SIAM Journal on Computing* 24.4 (1995), S. 873–921.

[42] Rodney G Downey und Michael Ralph Fellows. "Fixed-parameter tractability and completeness II: On completeness for W [1]". In: *Theoretical Computer Science* 141.1-2 (1995), S. 109–131.

[43] Rodney G Downey und Michael Ralph Fellows. "Parameterized computational feasibility". In: *Feasible mathematics II*. Springer, 1995, S. 219–244.

[44] Rodney G Downey und Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[45] Frank Drewes, Hans-Jörg Kreowski und Annegret Habel. "Hyperedge replacement graph grammars". In: *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*. World Scientific, 1997, S. 95–162.

[46] Bradley Efron. "Bootstrap methods: another look at the jackknife". In: *Breakthroughs in statistics*. Springer, 1992, S. 569–593.

[47] Hartmut Ehrig, Michael Pfender und Hans Jürgen Schneider. "Graph-grammars: An algebraic approach". In: *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*. IEEE. 1973, S. 167–180.

[48] Hans-Christian Ehrlich und Matthias Rarey. "Systematic benchmark of substructure search in molecular graphs-From Ullmann to VF2". In: *Journal of Cheminformatics* 4.1 (2012), S. 13.

[49] David Eppstein. "Parallel recognition of series-parallel graphs". In: *Information and Computation* 98.1 (1992), S. 41–55.

[50] Universität Erlangen. *Chemoinformatik*. 2019. URL: http://www2.chemie.uni-erlangen.de/projects/vsc/chemoinformatik/erlangen/index.html (besucht am 27. 05. 2019).

[51] Jerome Feder. "Plex languages". In: *Information Sciences* 3.3 (1971), S. 225–241.

[52] John Figueras. "Substructure search by set reduction". In: *Journal of Chemical Documentation* 12.4 (1972), S. 237–244.

[53] Pasquale Foggia, Carlo Sansone und Mario Vento. "A performance comparison of five algorithms for graph isomorphism". In: *Proceedings of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*. 2001, S. 188–199.

[54] Jerome Friedman, Trevor Hastie und Robert Tibshirani. *The elements of statistical learning*. Bd. 1. 10. Springer series in statistics New York, 2001.

[55] Michael R Garey und David S Johnson. *Computers and intractability: a guide to NP-completeness*. 1979.

[56] Johann Gasteiger u. a. *Handbook of chemoinformatics*. Bd. 1. Wiley Online Library, 2003.

[57] Joseph L Gastwirth. "The estimation of the Lorenz curve and Gini index". In: *The review of economics and statistics* (1972), S. 306–316.

[58] Chris Godsil und Gordon F Royle. *Algebraic graph theory*. Bd. 207. Springer Science & Business Media, 2013.

[59] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. (Besucht am 24. 01. 2019).

[60] Alex Graves, Abdel-rahman Mohamed und Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, S. 6645–6649.

[61] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke und Jürgen Schmidhuber. "A novel connectionist system for unconstrained handwriting recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2008), S. 855–868.

[62] Rozenberg Grzegorz, Ehrig Hartmut und Kreowski Hans-jorg. *Handbook Of Graph Grammars And Computing By Graph Transformations, Vol 3: Concurrency, Parallelism, And Distribution*. World Scientific, 1999.

[63] Ivan Gutman und Nenad Trinajstić. "Graph theory and molecular orbitals. Total φ-electron energy of alternant hydrocarbons". In: *Chemical Physics Letters* 17.4 (1972), S. 535–538.

[64] Annegret Habel. *Hyperedge replacement: grammars and languages*. Bd. 643. Springer Science & Business Media, 1992.

[65] Hermann Hager. *Hagers Handbuch der Pharmazeutischen Praxis: Band 8: Stoffe EO*. Springer-Verlag, 2013.

[66] Peter C Heinrich, Matthias Müller und Lutz Graeve. *Löffler/Petrides Biochemie und Pathobiochemie*. Springer-Verlag, 2014.

[67] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold u. a. "CNN architectures for large-scale audio classification". In: *2017 ieee international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2017, S. 131–135.

[68] Sepp Hochreiter und Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), S. 1735–1780.

[69] Alan L Hodgkin und Andrew F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), S. 500–544.

[70] R Webster Homer, Jon Swanson, Robert J Jilek, Tad Hurst und Robert D Clark. "SYBYL line notation (SLN): a single notation to represent chemical structures, queries, reactions, and virtual libraries". In: *Journal of chemical information and modeling* 48.12 (2008), S. 2294–2307.

[71] Harry B Hunt III, Robert L. Constable und Sartaj Sahni. "On the computational complexity of program scheme equivalence". In: *SIAM Journal on Computing* 9.2 (1980), S. 396–416.

[72] K Jetal Hunt, D Sbarbaro, R Żbikowski und Peter J Gawthrop. "Neural networks for control systems—a survey". In: *Automatica* 28.6 (1992), S. 1083–1112.

[73] Daylight Chemical Information Systems Inc. *Daylight Theory Manual*. 2011. URL: http://www.daylight.com/dayhtml/doc/theory/ (besucht am 30. 10. 2018).

[74] Daylight Chemical Information Systems Inc. *Simplified Molecular Input Line Entry System*. 2011. URL: https://daylight.com/smiles/ (besucht am 30. 10. 2018).

[75] Clay Mathematics Institute. *Millennium Prize Description and Rules*. URL: https://www.claymath.org/sites/default/files/millennium_prize_rules_0.pdf (besucht am 21. 06. 2019).

[76] John J Irwin und Brian K Shoichet. "ZINC- a free database of commercially available compounds for virtual screening". In: *Journal of chemical information and modeling* 45.1 (2005), S. 177–182.

[77] Nal Kalchbrenner, Edward Grefenstette und Phil Blunsom. "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188* (2014).

[78] Richard M Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, S. 85–103.

[79] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[80] Jon Kleinberg und Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[81] M Kor. *Single pushout transformations of generalized graph structures*. Techn. Ber. RP 220, Federal University of Rio Grande do Sul, Porto Alegre, Brazil, 1993.

[82] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2007. URL: http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf (besucht am 24. 01. 2019).

[83] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, S. 1097–1105.

[84] David Lagorce, Olivier Sperandio, Hervé Galons, Maria A Miteva und Bruno O Villoutreix. "FAF-Drugs2: free ADME/tox filtering tool to assist drug discovery and chemical biology projects". In: *BMC bioinformatics* 9.1 (2008), S. 396.

[85] Clemens Lautemann. "The complexity of graph languages generated by hyperedge replacement". In: *Acta Informatica* 27.5 (1990), S. 399–421.

[86] Victor Lavrenko, Raghavan Manmatha und Jiwoon Jeon. "A model for learning the semantics of pictures". In: *Advances in neural information processing systems*. 2004, S. 553–560.

[87] Steve Lawrence, C Lee Giles, Ah Chung Tsoi und Andrew D Back. "Face recognition: A convolutional neural-network approach". In: *IEEE transactions on neural networks* 8.1 (1997), S. 98–113.

[88] Andrew R Leach und Valerie J Gillet. *An introduction to chemoinformatics*. Springer Science & Business Media, 2007.

[89] Andy Liaw, Matthew Wiener u. a. "Classification and regression by randomForest". In: *R news* 2.3 (2002), S. 18–22.

[90] C. Loglisci. *Advanced Techniques for Mining Structured Data*. 2018. URL: http://www.di.uniba.it/~loglisci/PhDCourse/PhD_GraphMatching.pdf (besucht am 18. 03. 2019).

[91] Yvonne C Martin. "3D database searching in drug design". In: *Journal of Medicinal Chemistry* 35.12 (1992), S. 2145–2154.

[92] Oliver Mason und Mark Verwoerd. "Graph theory and networks in biology". In: *IET systems biology* 1.2 (2007), S. 89–119.

[93] Warren S McCulloch und Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), S. 115–133.

[94] Alan D McNaught und Alan D McNaught. *Compendium of chemical terminology*. Bd. 1669. Blackwell Science Oxford, 1997.

[95] U.S. National Library of Medicine. *PubChem*. 2019. URL: https://pubchem.ncbi.nlm.nih.gov/#query=CN(CO)SC&tab=substructure (besucht am 04. 06. 2019).

[96] Anukrati Mehta. *A Beginner's Guide to Classification and Regression Trees*. 2019. URL: https://www.digitalvidya.com/blog/classification-and-regression-trees/ (besucht am 02. 05. 2019).

[97] Yajie Miao, Mohammad Gowayyed und Florian Metze. "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding". In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE. 2015, S. 167–174.

[98] Joel C. Miller. *VF2 algorithm steps with example*. 2018. URL: https://stackoverflow.com/questions/8176298/vf2-algorithm-steps-with-example (besucht am 18. 03. 2019).

[99] Mehryar Mohri, Afshin Rostamizadeh und Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[100] Andreas C Müller, Sarah Guido u. a. *Introduction to machine learning with Python: a guide for data scientists*. Ö'Reilly Media, Inc.", 2016.

[101] Noel M O'Boyle, Michael Banck, Craig A James, Chris Morley, Tim Vandermeersch und Geoffrey R Hutchison. "Open Babel: An open chemical toolbox". In: *Journal of cheminformatics* 3.1 (2011), S. 33.

[102] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 10. 01. 2019).

[103] Theodosios Pavlidis. "Linear and context-free graph grammars". In: *Journal of the ACM (JACM)* 19.1 (1972), S. 11–22.

[104] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg u. a. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), S. 2825–2830.

[105] Karol J Piczak. "Environmental sound classification with convolutional neural networks". In: *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2015, S. 1–6.

[106] Krzysztof Pietrzak. "On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems". In: *Journal of Computer and System Sciences* 67.4 (2003), S. 757–771.

[107] Günter Daniel Rey und Karl F Wender. *Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Huber, 2010.

[108] Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[109] Grzegorz Rozenberg. *Handbook of Graph Grammars and Comp.* Bd. 1. World scientific, 1997.

[110] Grzegorz Rozenberg und Arto Salomaa. *Handbook of Formal Languages: Volume 3 Beyond Words*. Springer Science & Business Media, 2012.

[111] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[112] Thomas J Schaefer. "The complexity of satisfiability problems". In: *Proceedings of the tenth annual ACM symposium on Theory of computing*. ACM. 1978, S. 216–226.

[113] Michael Schröder. "Einführung in die kurzfristige Zeitreihenprognose und Vergleich der einzelnen Verfahren". In: *Prognoserechnung*. Springer, 2012, S. 11–45.

[114] Harry P Schultz. "Topological organic chemistry. 1. Graph theory and topological indices of alkanes". In: *Journal of Chemical Information and Computer Sciences* 29.3 (1989), S. 227–228.

[115] Ferdi Schüth. "Heterogene Katalyse. Schlüsseltechnologie der chemischen Industrie". In: *Chemie in unserer Zeit* 40.2 (2006), S. 92–103.

[116] Gordon M Shepherd. *The synaptic organization of the brain*. Oxford University Press, 2003.

[117] Julian R Ullmann. "An algorithm for subgraph isomorphism". In: *Journal of the ACM (JACM)* 23.1 (1976), S. 31–42.

[118] Stephan M Wagner und Nikrouz Neshat. "Assessing the vulnerability of supply chains using graph theory". In: *International Journal of Production Economics* 126.1 (2010), S. 121–129.

[119] Ingo Wegener. *Komplexitätstheorie: Grenzen der Effizienz von Algorithmen*. Springer-Verlag, 2013.

[120] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of chemical information and computer sciences* 28.1 (1988), S. 31–36.

[121] David Weininger, Arthur Weininger und Joseph L Weininger. "SMILES. 2. Algorithm for generation of unique SMILES notation". In: *Journal of Chemical Information and Computer Sciences* 29.2 (1989), S. 97–101.

[122] Sholom M Weiss, Ioannis Kapouleas und JW Shavlik. "An empirical comparison of pattern recognition, neural nets and machine learning classification methods". In: *Readings in machine learning* (1990), S. 177–183.

[123] Peter Willett. "Similarity-based virtual screening using 2D fingerprints". In: *Drug discovery today* 11.23-24 (2006), S. 1046–1053.

[124] Ian H Witten, Eibe Frank, Mark A Hall und Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[125] Jun Xu und Arnold Hagler. "Chemoinformatics and drug discovery". In: *Molecules* 7.8 (2002), S. 566–600.

[126] Ji Yang. *ReLU and Softmax Activation Functions*. 2017. URL: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions (besucht am 05. 03. 2019).

[127] Derek York. "Least-squares fitting of a straight line". In: *Canadian Journal of Physics* 44.5 (1966), S. 1079–1086.

[128] Antonio Zamora. "An algorithm for finding the smallest set of smallest rings". In: *Journal of Chemical Information and Computer Sciences* 16.1 (1976), S. 40–43.

[129] DECHEMA e.V. *Fullerene – Materialinfo*. 2019. URL: https://www.nanopartikel.info/nanoinfo/materialien/fullerene/materialinfo-fullerene (besucht am 22. 07. 2019).