Universität Siegen

Fachbereich 12 - Elektrotechnik und Informatik

Diplomarbeit

Optimierung des SiDiff-Algorithmus unter Ausnutzung modellspezifischer Eigenschaften und Strukturen

Pit Pietsch

Erstprüfer: Prof. Dr. Kelter, Universität Siegen
Zweitprüfer: Prof. Dr. Fathi, Universität Siegen

Siegen, November 2008

Zusammenfassung

Die Differenzanalyse von technischen Dokumenten besitzt in ihrem Kern eine quadratische Laufzeit. Auf die reale Laufzeit wirkt sich allerdings auch die Komplexität der Vergleichsfunktionen aus, mit denen die Berechnung der Ähnlichkeitswerte zwischen zwei Elementen erfolgt. In dieser Arbeit wird geprüft, ob ein Verfahren aus der Bioinformatik, das fingerprintbasierte Screening, dazu geeignet ist, schnell die Menge an Elementen zu bestimmen, die eine gute Aussicht auf hohe Ähnlichkeit zu einem Musterelement besitzen. Auf diese Weise soll der Aufwand für die Berechnung des Ähnlichkeitswertes zwischen Elementen mit wenig Gemeinsamkeit gespart werden. Wie gezeigt wird, lässt sich die reale Laufzeit der Differenzanalyse mitunter drastisch verkürzen. Allerdings sind die erzielten Ergebnisse kritisch zu sehen, da das Verfahren in seinem neuen Anwendungskontext einzig ein heuristischer Ansatz ist, der keine Garantie auf eine Berechnung optimaler Ergebnisse geben kann.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung und Motivation

In der heutigen Zeit besitzen Projekte und Geschäftsprozesse einen stetig steigenden Komplexitätsgrad. Deshalb ist es wichtig, den Überblick über alle dabei anfallenden Dokumente zu behalten. Dies lässt sich einzig durch eine angemessene Versionierung und Archivierung der Dokumente gewährleisten. Gerade unter diesem Gesichtspunkt stellen technische Dokumente eine besondere Herausforderung dar. Ein grundsätzliches Merkmal technischer Dokumente ist, dass sie in ihrem Lebenszyklus eine stetige Entwicklung durchlaufen. Zudem ist es für größere Projekte nicht unüblich, dass in mehreren Entwicklungszweigen gleichzeitig gearbeitet wird und somit nicht nur eine einzelne und zeitlich klar nachzuvollziehende Versionshistorie vorliegt, sondern mehrere nebenläufige Versionen eines Dokumentes existieren.

Anwendungsgebiete, in denen derartige technische Dokumente entstehen, sind beispielsweise die Softwaretechnik mit den verschiedenen Diagrammtypen aus der *Unified Modeling Language* (*UML*) oder die Ingenieurswissenschaften mit der Modellierung von dynamischen Systemen in MATLAB/Simulink-Diagrammen. Für den jeweiligen Nutzer der Dokumente ist es für seine tägliche Arbeit oftmals entscheidend, sich über die Unterschiede und Gemeinsamkeiten zwischen den verschiedenen Versionen im Klaren zu sein. Eine Aufgabe, die selbst bei Projekten einer moderaten Größenordnung nicht ohne die Unterstützung von Software gelöst werden kann.

Für textbasierte technische Dokumente stehen dem Anwender dabei eine Vielzahl an Lösungen zur Verfügung, die ihn bei der Durchführung einer Differenzanalyse unterstützen. Für nicht-textbasierte Dokumente - wie sie die bereits angesprochenen Dokumenttypen repräsentieren - ist die Zahl der angebotenen Lösungen deutlich überschaubarer.

Nicht-textuelle technische Dokumente weisen dabei in der Regel einen hohen Grad an Strukturiertheit auf. Im Gegensatz zu textbasierten Dokumenten bestehen sie nicht einfach aus einer Ansammlung von Zeilen und Zeichen, sondern lassen sich als eine Ansammlung von Elementen verschiedener Typen beschreiben. Die einzelnen Elemente können dabei untereinander sowohl in hierarchischen als auch heterarchischen Beziehungen stehen.

Ein erster Ansatz zur Berechnung von Differenzen zwischen zwei Dokumenten beruht auf der Durchführung eines paarweisen Vergleichs. Dabei ist die Ähnlichkeit eines jeden Elements zu jedem weiteren Element aus dem zu vergleichenden Dokument, aufgeschlüsselt nach Elementtypen, zu berechnen. Die Problematik dieses Ansatzes liegt in der impliziten quadratischen Laufzeit. Selbst für relativ kleine Elementmengen ist eine große Anzahl an Vergleichen durchzuführen. Für den Fall, dass in zwei Dokumenten jeweils 25 Elemente eines Typs vorkommen, sind bereits $25 * 25 = 625$ Direktvergleiche notwendig. Bedenkt man, dass in einigen Modellen Elemente eines Typs mitunter in drei- oder vierstelliger Anzahl auftreten, wird das Ausmaß dieser Problematik leicht ersichtlich. Erfolgt die Berechnung eines Ähnlichkeitswertes zwischen Elementen weiterhin auf Basis mehrerer, aufwendig zu berechnender Vergleichskriterien, so wird die vorliegende Situation zusätzlich verschärft.

Problemstellung

Es stellt sich die Frage, ob ein exakter Vergleich aufgrund komplexer Vergleichskriterien effizient ist. Lässt sich anhand einfacher Kriterien bereits vor einer genauen Berechnung der Ähnlichkeit abschätzen, ob zwei Elemente später eine hohe oder niedrige Ähnlichkeit zueinander aufweisen, so kann dieses Wissen zu einer Optimierung des Verfahrens genutzt werden. Der Grundgedanke besteht darin, mittels schneller Abschätzungen zu entscheiden, ob zwei Elemente genügend Ähnlichkeit aufweisen, um eine genaue Berechnung ihrer Ähnlichkeit zu rechtfertigen. Somit werden aufwendige Berechnungen nur noch für Elementpaare vorgenommen, die auch eine gute Aussicht auf eine hohe Ähnlichkeit aufweisen.

Ziel dieser Arbeit

Ein Verfahren aus der Domäne der Bioinformatik, das *fingerprintbasierte Screening*, löst eine ähnliche Problemstellung. In seinem originären Kontext dient es der Optimierung von Suchanfragen auf molekulare Datenbanken. Anstatt anhand aufwendiger Vergleichsfunktionen für jedes einzelne Molekül zu entscheiden, ob es auf die Suchanfrage passt, wird in einem ersten Schritt auf Basis einfacher Kriterien die Menge an Molekülen gefiltert, die definitiv nicht auf die Suchanfrage zutreffen. Einzig für die restlichen Moleküle ist dann im Weiteren zu bestimmen, ob sie auf die Suchanfrage passen.

Um schnell eine Selektion der Kandidatenmenge vorzunehmen, wird dabei insbesondere auf einen *Fingerprint* zurückgegriffen. Ein Fingerprint ist eine Bitfolge, die charakteristische und repräsentative Eigenschaften eines Moleküls codiert.

Ziel der Arbeit ist ein Transfer des aus der Bioinformatik stammenden Konzepts eines fingerprintbasierten Screenings in den Kontext der Differenzanalyse von Dokumenten. Dabei soll insbesondere die Vergleichsphase optimiert werden, indem anhand einer schnellen Abschätzung entschieden wird, ob eine genaue Berechnung eines Ähnlichkeitswerts zwischen zwei Elementen als sinnvoll anzusehen ist.

Die Schwierigkeit dieser Aufgabe liegt in den Unterschieden zwischen den Einsatzgebieten, sowie der Problematik einer Abbildung von den charakteristischen Kriterien eines Elements auf eine geeignete binäre Codierung. Weiterhin ist zu untersuchen, auf Basis welcher Kriterien sich schnell entscheiden lässt, ob zwei Elemente bei einer genauen Untersuchung eine hohe Ähnlichkeit aufweisen.

Gliederung der Arbeit

Kapitel 2 beschäftigt sich eingangs näher mit dem *Sequence Alignment*, einer weiteren Problemstellung aus dem Anwendungsgebiet der Bioinformatik. Es ist für einige Aufgabenstellungen dabei von Interesse, die Ähnlichkeit zwischen zwei biologischen Sequenzen auf Basis ihrer optimalen Ausrichtung zu ermitteln. Im Laufe des Kapitels wird gezeigt, warum klassische Verfahren für eine Ähnlichkeitsberechnung zwischen Zeichenketten unter den gegebenen Rahmenbedingungen oftmals nicht ausreichen und welche Algorithmen in der Bioinformatik für die Lösung dieser Problemstellung genutzt werden.

In Kapitel 3 wird die Motivation für den Einsatz einer fingerprintbasierten Selektion von Kandidatenmengen erklärt. Danach werden zwei Ansätze zur Erstellung von Fingerprints vorgestellt. Im Anschluss wird dargestellt, welche Möglichkeiten für die Berechnung einer Ähnlichkeit zwischen zwei Fingerprints existieren. Ein derart berechneter Ähnlichkeitswert ist die Grundlage für eine weitere Selektion der Elemente.

In Kapitel 4 wird zuerst ein Algorithmus für die Differenzbildung zwischen zwei technischen Dokumenten vorgestellt. Danach wird aufgezeigt, in wie fern der Einsatz eines fingerprintbasierter Screening-Ansatzes in dem Rahmen der Differenzanalyse von Interesse ist. Dabei wird ausführlich auf die bei dem Transfer aufgetretenen Problematiken eingegangen. Weiterhin wird gezeigt, welche Eigenschaft ein Modell aufweisen muss, um für den Einsatz eines Fingerprintverfahrens geeignet zu sein. Abschließend wird die Implementierung des Fingerprintverfahrens für die Differenzanalyse skizziert.

Die Evaluation der vorgestellten Implementierung erfolgt in Kapitel 5. Dazu werden die Modelltypen *UML-Klassendiagramm*, *UML-Aktivitätsdiagramm* sowie *MATLAB/Simulink-Modelle* mittels verschiedener Ansätze für die Abbildung charakteristischer Merkmale in einem Fingerprint auf ihre Eignung für das vorgestellte Verfahren untersucht.

Die Arbeit schließt mit Kapitel 6, wo die Ergebnisse der Arbeit repetiert, sowie weitere Fragestellungen und Einsatzgebiete in Zusammenhang mit Fingerprints aufgezeigt werden.

2. Algorithmen in der Bioinformatik

Das folgende Kapitel gibt einen Überblick über die in der Bioinformatik häufig auftretende Problemstellung zwei Sequenzen anhand einer gegebenen Bewertungsfunktion optimal zueinander auszurichten (engl. *Sequence Alignment)*. Dabei wird genauer untersucht warum klassische Verfahren aus der Informatik, die sich mit dem verwandten Problem der Ähnlichkeitsbestimmung zwischen zwei Zeichenketten beschäftigen, für die vorliegende Fragestellung keine zufriedenstellende Lösung liefern. Weiterhin werden mit den Verfahren von *Needleman-Wunsch* und *Smith-Waterman* zwei in der Bioinformatik verbreitete Algorithmen vorgestellt, mit denen sich entsprechende Lösungen ermitteln lassen.

2.1 Sequence Alignment

Ein *Alignment* (dts. *Ausrichtung*) ist ein Vergleich zweier Sequenzen, wobei eine *Sequenz* in diesem Zusammenhang als eine Aneinanderreihung von Zeichen (oder auch: *Zeichenkette*) eines gegebenen Alphabets definiert ist. Besonders häufig werden Alignments dabei im Kontext der Bioinformatik eingesetzt, beispielsweise um verschiedene, innerhalb von Experimenten ermittelte, genetische Sequenzen (siehe [Böc07], S. 81) zu vergleichen.

Sequence Alignement beschäftigt sich mit der Fragestellung, wie zwei (oder mehrere) gegebene Zeichenketten (im Folgenden wird der Begriff Zeichenkette synonym mit dem Begriff Sequenz genutzt) sich so zueinander ausrichten lassen, dass ähnliche Abschnitte innerhalb der Sequenzen erkennbar werden. Eine Identifikation ähnlicher Teilbereiche kann dabei helfen, Rückschlüsse auf die jeweiligen funktionellen, evolutionären oder strukturellen Eigenschaften der zugrundeliegenden Sequenzen zu ziehen. Die klassischen Anwendungsfelder für diese Fragestellung sind Vergleiche von Proteinketten[1] und DNA-Sequenzen[2].

[1] *Proteine, oder auch Eiweiße, sind die Grundbausteine aller Zellen. Ein Protein wird dabei aus einer Kette von Aminosäuren gebildet. Bei Menschen kann zwischen 20 verschiedenen Aminosäuren*

Alignments werden häufig zur Bestimmung einer evolutionären Verwandtschaft zwischen verschiedenen Sequenzen eingesetzt. Die Grundidee hinter diesem Ansatz (siehe [Sel04], S. 71) ist, dass die Natur nicht für jedes Lebewesen eine eigene Biologie entwickelt, sondern ein einmal bewährtes Allgemeinkonzept mit der Zeit nur weiter verbessert und gegebenenfalls an Umweltbedingungen adaptiert. Werden dementsprechend Sequenzen gefunden, die eine relative hohe Ähnlichkeit aufweisen, dann lassen sich daraus Rückschlüsse auf deren Verhalten und Funktion ziehen.

```
Sequenz1:     A-GATTACTCAA--TTCCTA
Sequenz2:     -BGATTA-TCAAGATTCCTG
```

Abbildung 1: Alignment zweier DNA-Sequenzen

Auf den ersten Blick weist die vorliegende Problemstellung eine hohe Ähnlichkeit zu der in der klassischen Informatik gut erforschten Problemstellung des Vergleichs von zwei Strings auf. Ein *String* ist letztendlich, genau wie eine Sequenz[3], als eine aneinandergefügte Kette aus Zeichen eines zugrundeliegenden Alphabets definiert. Es existieren aber trotz der augenscheinlichen Ähnlichkeit in der Problematik in einem biologischen Kontext einige Besonderheiten, so dass bewährte Verfahren für den Vergleich zweier Zeichenketten nicht ohne weiteres übertrag- und anwendbar sind.

Klassische Verfahren wie beispielsweise die zur Ermittlung der Unterschiedlichkeit zwischen Zeichenketten oft genutzte und von *Robert Wesley Hamming* 1950 in dem Paper *error-detecting and error-correcting codes* [Ham50] vorgestellte *Hamming-Distanz*[4], oder aber auch die Berechnung des *Longest Common Substrings* (siehe [Bla04] für eine Implementierung) greifen für die vorliegende Aufgabenstellung zu kurz.

---

unterschieden (vgl. [Böc07], S. 8) werden, was selbst bei relativ kleinen Ketten eine enorme Anzahl an denkbaren Verknüpfungsmöglichkeiten ermöglicht.

[2] Die DNA ist die Trägerin der Erbgutinformationen aller Lebewesen und DNA-Viren. Ihre chemische Struktur (siehe [Sel04], S. 35) ist die eines Kettenmoleküls aus Nukleotiden, die gemäß des Anfangsbuchstabens ihrer Base mit G, T, C oder A dargestellt werden.

[3] Der Grund für die unterschiedliche Begrifflichkeit (siehe [Böc07], S.81) ist deutlich zu machen, dass im Kontext eines Alignments zweier Zeichenketten Subsequenzen eine wichtigere Rolle spielen als Substrings.

[4] Die Hamming-Distanz wird später in dem Kapitel 3.4 Ähnlichkeitsberechnung zwischen Fingerprints nochmals aufgegriffen.

Besonderheiten, die bei einem Alignment der betrachteten Sequenzen aus der Biologie auftreten (siehe[Böc07], S.19) und in den genannten klassischen Verfahren keine Berücksichtigung finden sind:

- Substitutionen sowie
- Einfügungen und Löschungen.

Substitutionen

Eine Substitution (engl. *Substitution*) liegt vor, wenn einem Zeichen aus einer Sequenz ein nicht identisches Zeichen der zu vergleichenden Sequenz zugeordnet ist. In Abbildung 1 ist beispielhaft ein Alignment zweier DNA-Sequenzen, die sich in weiten Teilen gleichen, dargestellt. Wie an der letzten Stelle der Ausrichtung zu beobachten ist, kam es dort zu einer Substitution der *Base A* aus *Sequenz 1* mit der *Base G* aus *Sequenz 2*. In der Biologie wird für das Auftreten einer Substitution auch oftmals der Begriff *Mutation* verwendet. Das Auftreten von Mutationen ist dabei in der Natur nicht ungewöhnlich, sind sie doch als Basis der evolutionären Entwicklung aller Lebewesen zu verstehen.

Einfügungen und Löschungen

Neben Substitutionen ist in biologischen Sequenzen oftmals zu entdecken, dass im Laufe ihrer evolutionären Entwicklung ein Teil einer Sequenz gelöscht (engl. *Deletion*) oder ein neues Sequenzstück eingefügt wird (engl. *Insertion*). Einfüge- und Löschoperationen führen zu Lücken, sogenannten *Gaps,* bei der Darstellung eines Alignments zweier Sequenzen. Ein Gap ist innerhalb einer Ausrichtung definiert als eine fortlaufende Sequenz an Leerräumen, wobei in der Regel jedes Zeichen durch den sogenannten *Gap-Character* "-" symbolisiert wird. In Abbildung 1 sind diese Sachverhalte innerhalb der ersten beiden Stellen des Alignments nachzuvollziehen, wobei an der ersten Position ein Löschen der Base A aus *Sequenz 2* stattfindet, während an der nächsten Position der *Sequenz 2* das Einfügen der Base B zu erkennen ist.

Zieht man diese besonderen Eigenschaften mit in Betracht, so ist es leicht ersichtlich, dass die bereits erwähnten klassischen Verfahren zur Bestimmung der Ähnlichkeit (bzw. Unterschiedlichkeit) vor diesem Hintergrund an ihre Grenzen stoßen. Ein Vergleich der DNA-Kette *GCGCGCG* mit der DNA-Kette *CGCGCGC* nach dem von *Hamming* vorgeschlagenen Distanzmaß ergibt beispielsweise, dass

die beiden Sequenzen komplett unterschiedlich sind. Es ist allerdings mit bloßem Auge sofort zu erkennen, dass sich die beiden Sequenzen sehr wohl ähneln (vgl. [Jon04], S. 167). Das Problem liegt in der impliziten Annahme der Hamming-Distanz, dass bei jedem Vergleich die zu vergleichenden Zeichenketten schon passend zueinander ausgerichtet sind, also das *i*-te Symbol der ersten Sequenz mit dem *i*-ten Symbol der zweiten Sequenz korrespondiert[5]. Auch ein Verfahren zum Finden des längsten, gemeinsamen Substrings kann bei der gegebenen Aufgabenstellung nicht zu einem zufriedenstellenden Ergebnis führen: Eine an sich zusammenhängende Teilsequenz kann an einer Stelle durch das Löschen, Einfügen oder Substituieren eines einzelnen Zeichens oder eines ganzes Sequenzteils unterbrochen sein.

Ein erster Ansatz, der dazu genutzt werden kann die Ähnlichkeit zweier biologischer Sequenzen zu beschreiben, ist die 1965 von *Levenshtein* (siehe [Bla08] für eine Implementierung) vorgestellte *Edit-Distanz*. Die Edit-Distanz kann dabei als eine Generalisierung der Hamming-Distanz aufgefasst werden, die es weiterhin ermöglicht auch Sequenzen verschiedener Länge zu vergleichen (siehe [Jon04], S. 176). Neben den von *Hamming* vorgeschlagenen Operationen für ein Ersetzen von Zeichen berücksichtigt *Levenshtein* auch Operationen für das Einfügen und Löschen einzelner Zeichen. Untersucht man die Kostenfunktion die der Edit-Distanz zu Grunde liegt, so stellt man fest, dass für jede der Edit-Operationen ein Kostenfaktor von *1* angelegt ist, während eine Übereinstimmung von Zeichen kostenneutral ist und den Wert *0* aufweist.

Die Bestimmung der Edit-Distanz stellt sich demnach als Minimierungsproblem dar, wobei die Ähnlichkeit zweier Sequenzen als die minimale Anzahl an Edit-Operationen zu verstehen ist, die nötig sind, um *Sequenz 1* in *Sequenz 2* zu überführen. Je mehr Edit-Operationen für die entsprechende Transformation benötigt werden, umso weiter entfernt voneinander sind die Sequenzen anzusehen. Die in dem vorletzten Absatz beispielhaft aufgeführten DNA-Ketten weisen eine Edit-Distanz von *2* auf. Es ist in *Sequenz 1* einzig das erste Zeichen zu löschen, sowie an letzter Stelle ein weitere *Base G* einzufügen, um sie in *Sequenz 2* zu überführen. Dabei ermöglicht die vorgestellte *Edit-Distanz* es nicht nur Zeichenketten auf Basis ihrer Zeichen zu untersuchen, sondern es lassen sich - je nach Anwendungskontext - auch Wörter oder Sätze als einzelne Einheiten des zugrundeliegenden Alphabets auffassen (siehe [Mas03], S. 27).

Ein weiteres einfaches Verfahren zur Untersuchung der Ähnlichkeit von Sequenzen ist das Finden der längsten gemeinsamen Subsequenz (engl. *Longest Common Subsequence, LCS*)[6], welches in [Cor07] ab S. 151 näher beschrieben ist.

---

[5] *Ein weiteres Problem liegt darin, dass bei einem Vergleich mittels Hamming-Distanz von gleichlangen Sequenzen ausgegangen wird. Dies ist allerdings bei DNA- und Protein-Sequenzen nicht zwingend gegeben.*

[6] *Für eine Implementierung des Algorithmus siehe [Bla06]*

Die Suche einer LCS zwischen zwei Sequenzen wird auf Basis ihres *Edit-Graph* (siehe auch [Weh04], S.13ff) durchgeführt.

Bei der Bestimmung einer LCS (vgl. [Jon04], S. 172) werden keine Substitutionen berücksichtigt, sondern es ist nur noch das Einfügen und Löschen von Zeichen erlaubt. Dieser Sachverhalt kann als eine Veränderung der Kostenfunktion interpretiert werden, bei der Substitutionen mit höheren Kosten verbunden sind als Einfüge- und Lösch-Operationen. Somit ist das Auftreten einer Substitution zwischen zwei Zeichen in einer optimalen Lösung nicht mehr möglich und kann in letzter Konsequenz folglich auch bei deren Ermittlung vernachlässigt werden.

Wird beispielsweise eine LCS der beiden DNA-Sequenzen *TAGTAT* und *TGCAT* gesucht, ergibt sich nach Anwendung des von *Wehren* beschriebenen Algorithmus als Lösung eine Subsequenz der Länge 4, nämlich *TGAT*. Abbildung 2 zeigt dabei die der Lösung entsprechende Ausrichtung der beiden Sequenzen. Dabei sind, um das vorliegende Alignment der Sequenzen zu bilden, eine Lösch-, sowie zwei Einfüge-Operationen nötig.

```
Sequenz 1:    TAGT-AT
Sequenz 2:    T-G-CAT
```

Abbildung 2: Alignment mittels LCS

Einfach ausgedrückt ist ein durch LCS ermitteltes Alignment zweier Sequenzen die maximale Anzahl möglicher Übereinstimmungen, welche zwischen den einzelnen Sequenzzeichen gebildet werden kann, ohne dabei deren Reihenfolge zu verändern.

Eine Bewertungsfunktion, wie sie in dem Kontext der *Longest Common Subsequence* Anwendung findet, greift allerdings für die Problemstellung des Sequence Alignments in der Biologie zu kurz. Hauptgrund hierfür ist, dass die bei LCS vernachlässigten Substitutionen von Zeichen durchaus vorkommen und darüber hinaus auch nicht beliebig auftreten. So gibt es beispielsweise Aminosäuren, die sich in ihrer Funktion (vgl. [Jon04], S. 178) ähneln und deren Substitutionen dementsprechend häufig in Proteinketten zu finden sind. Andere Substitutionen von Aminosäuren, welche nur wenig Gemeinsamkeit in Struktur und Funktion aufweisen, sind hingegen nur äußerst selten zu beobachten. Eine Bewertungsfunktion, welche Substitutionen einzig mit einem fixen Kostenfaktor berücksichtigt - oder wie LCS komplett von ihnen abstrahiert - gestaltet sich unter diesem Gesichtspunkt als unzureichend. Stattdessen sollte eine Substitution von niedriger Wahrscheinlichkeit mit höheren Kosten verbunden und somit für ein

Alignment unattraktiver sein, als Substitutionen die mit höherer Wahrscheinlichkeit vorzufinden sind.

Wie in diesem Abschnitt gezeigt wurde, reichen die vorgestellten klassischen Verfahren und Algorithmen des Sequenzvergleichs nicht aus, um die besonderen Aspekte der Problemstellung des Sequenzvergleichs in einem biologischen Kontext zu berücksichtigen. Gesucht sind im Weiteren folglich Bewertungsfunktionen, auf deren Basis eine Evaluation eines gegebenen Alignments zweier biologischer Sequenzen sinnvoll erfolgen kann. Im nächsten Abschnitt wird ein Ansatz einer derartigen Bewertungsfunktion vorgestellt und mit *Ähnlichkeitsmatrizen* ein weiteres Hilfsmittel für die Bewertung von Alignments eingeführt. In den darauffolgenden Abschnitten werden zwei Algorithmen vorgestellt, die ein optimales Alignment unter jeweils unterschiedlichen Gesichtspunkten ermitteln.

Bewertung des Alignments zweier biologischer Sequenzen

Um den vorgestellten Anforderungen, welche aus Substitutionen unterschiedlicher Wahrscheinlichkeiten resultieren, gerecht zu werden, wurden in der Bioinformatik *Ähnlichkeitsmatrizen* entwickelt. Eine Ähnlichkeitsmatrix weist dabei den verschiedenen möglichen Paarungen der Zeichen des vorliegenden Alphabets Werte zu, die widerspiegeln, wie gut das entsprechende Alignment der beiden Zeichen angesehen wird.

Die folgenden Beispiele erfolgen anhand von DNA-Sequenzen[7]. Dabei wird in der Regel einer Übereinstimmung zweier Basen ein positiver Wert zugewiesen, während Substitutionen negative Werte zugeordnet sind. Je niedriger die angenommene Wahrscheinlichkeit einer Substitution der Basen, desto niedriger ist der abgebildete Wert. Es ist leicht ersichtlich, dass die resultierende Matrix symmetrische Eigenschaften aufweist.

Weiterhin wird neben einer Ähnlichkeitsmatrix noch ein fester Kostenfaktor für *Gaps* (das sogenannte *Gap-Penalty*) eingeführt. Ein Gap-Penalty bestraft das Auftreten der durch Einfüge- und Löschoperationen entstehenden Lücken die in einer gegebenen Ausrichtung zweier Sequenzen auftreten.

---

[7] *In der Praxis sind Ähnlichkeitsmatrizen bei einem Vergleich von Proteinketten geläufiger als bei einem Vergleich von DNA-Sequenzen. Für DNA-Sequenzen wird oftmals von vereinfachten Bedingungen ausgegangen und einzig die Kosten für auftretende Lücken, sowie die Kosten für eine Fehlzuordnung definiert (siehe [Jon04], S.178). Aus Gründen der Übersichtlichkeit wird das Prinzip hier an DNA Sequenzen gezeigt. Ähnlichkeitsmatrizen für Proteinketten sind auf Grund des zugrundeliegenden Alphabets (20 Aminosäuren) sehr umfangreich. Das Prinzip lässt sich natürlich ohne Änderungen auch auf einen Vergleich von Proteinketten übertragen.*

Abbildung 3 zeigt eine Ähnlichkeitsmatrix δ für DNA. Für das folgende Beispiel sei weiterhin ein *Gap*-Penalty k mit einem Wert von $k = -6$ festgelegt. Um mittels der Ähnlichkeitsmatrix δ auch eine Bewertung von auftretenden Lücken zu ermöglichen, wurde das zugrundeliegende Alphabet um den Gap-Character "-" erweitert und die jeweiligen Kosten ebenfalls in der Ähnlichkeitsmatrix δ abgebildet.

	A	**G**	**C**	**T**	-
A	8	-3	-10	-1	-6
G	-3	10	-6	-5	-6
C	-10	-6	9	-2	-6
T	-1	-5	-2	7	-6
-	-6	-6	-6	-6	0

Abbildung 3: Ähnlichkeitsmatrix δ

Wird nun eine beliebige Ausrichtung zweier Sequenzen betrachtet, so kann einer Zuordnung zweier Zeichen w und x des zugrundeliegenden Alphabets innerhalb des Alignments mit Hilfe der Ähnlichkeitsmatrix über die Funktion $\delta(w, x)$ ein Wert zugewiesen werden, der die Qualität der vorliegenden Paarung ausdrückt. Die Summe der Werte über die einzelnen Zuordnungen einer Ausrichtung drückt dabei deren Güte aus. Im Weiteren kann eine auf diesem Weg gewonnene Bewertung genutzt werden, verschiedene Alignments in Relation zu setzen.

Betrachtet man beispielsweise die in Abbildung 2 mittels LCS gefundene Ausrichtung unter den gegebenen Bedingungen, so ergibt sich für die Güte des Alignments folgender Wert:

$$\delta(T,T) + \delta(A,-) + \delta(G,G) + \delta(T,-) + \delta(-,C) + \delta(A,A) + \delta(T,T)$$
$$= 7 - 6 + 10 - 6 - 6 + 8 + 7$$
$$= 14$$

Untersucht man andere Ausrichtungen der beiden Sequenzen, so ergeben sich trivialerweise mitunter abweichende Werte.

In diesem Abschnitt wurde ein Ansatz aufgezeigt, auf dessen Basis ein Wert ermittelt werden kann, der die Güte eines Alignments ausdrückt. Der dabei berechnete Wert kann für einen Vergleich der Qualität verschiedener Alignments genutzt werden. Dabei stellt sich natürlich die Frage, wie sich algorithmisch ein Alignment ermitteln lässt, dessen Ausrichtung unter einer gegebenen Bewertung als optimal anzusehen ist. In der Bioinformatik existieren dabei zwei verschiedene Fragestellungen die es zu beantworten gilt:

- Was ist das optimale Alignment zweier Sequenzen über deren gesamte Länge (*Global Alignment Problem*)?
- Was ist das optimale Alignment, dass über Teile zweier Sequenzen gebildet werden kann (*Local Alignment Problem*)?

In den folgenden Abschnitten wird auf beide Fragestellungen eingegangen. Dabei werden die unterschiedlichen Ansätze genauer beschrieben. Weiterhin werden Algorithmen vorgestellt, auf deren Basis sich Lösungen für die jeweilige Aufgabenstellung berechnen lassen.

2.2 Global Alignment Problem

Das Problem, ein optimales Alignment zweier Sequenzen über deren gesamte Länge zu ermitteln, ist in der Literatur als *Global Alignment Problem* bekannt. Es wird von *Jones* (siehe [Jon04], S. 177) wie folgt definiert:

> ***Global Alignment Problem:***
>
> *Find the best global alignment between two strings under a given scoring matrix.*
>
> *Input: Strings v und w and a scoring matrix δ*
>
> *Output: An alignment of v and w whose score (as defined by the matrix δ) is maximal among all possible alignments of v and w.*

Ein Algorithmus zur Lösung der gegebenen Fragestellung wird in dem folgenden Abschnitt vorgestellt.

Needleman-Wunsch-Algorithmus

Ein Algorithmus der zum Finden eines optimalen globalen Alignments zweier Sequenzen eingesetzt werden kann, ist in dem von *Saul Needleman* und *Christian Wunsch* im Jahr 1970 veröffentlichten Artikel *A general method applicable to the search for similarities in the amino acid sequence of two proteins* (siehe [Nee70]) beschrieben und im Weiteren nach den beiden Erstellern benannt. Der Algorithmus greift dabei auf Methoden der dynamischen Programmierung zurück, indem er die Lösung des vorliegenden Problems auf die Lösung kleinerer Teilprobleme reduziert (vgl. [Böc07], S. 84). Für die Bewertung der einzelnen Zuordnungen wird eine um den Gap-Character erweiterte Ähnlichkeitsmatrix, wie sie im letzten Kapitel vorgestellt wurde, verwendet.

Der Algorithmus erstellt dazu als erstes eine Matrix der Größe M($n+1$, $m+1$), wobei n die Länge der ersten Sequenz und m die Länge der zweiten Sequenz repräsentiert. Die Matrix ist um jeweils eine Spalte und Zeile größer als die Länge der Sequenzen um die Bildung der durch Einfüge- und Löschoperationen entstehenden Lücken am Anfang - respektive am Ende - des Alignments zu

ermöglichen. Jeder Matrixeintrag wird mittels der gegebenen Ähnlichkeitsmatrix δ und der in Abbildung 4 dargestellten Formel berechnet.

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + \delta(xi, yj) \\ M(i-1,j) - \delta(xi, -) \\ M(i,j-1) - \delta(-, yi) \end{cases}$$

Abbildung 4: Formel globales Alignment

Da nicht von Anfang an die für eine Berechnung nötigen Werte für alle Felder zur Verfügung stehen, werden die Einträge sukzessiv ausgehend von Feld *M(2,2)* berechnet. Weiterhin wird sich dabei mittels Zeigern gemerkt, auf Basis welches (oder welcher) der drei in den Formeln genutzten Ursprungsfelder ein jeweiliger Eintrag berechnet wurde. Sind auf diese Weise alle Einträge der Matrix berechnet, lässt sich die Güte des optimalen Alignments der beiden Sequenzen in dem Feld *M(n+1, m+1)* ablesen.

Angefangen von dem in *M(n+1, m+1)* errechneten optimalen Wert für die gesamte Ausrichtung der beiden Sequenzen wird nun mittels Backtracking das optimale Alignment bestimmt. Dazu wird einfach der bei der Berechnung genommene und mittels Zeigern gemerkte Weg bis zu dem Feld M(*1,1*)=0 zurückverfolgt und die Sequenzen dementsprechend ausgerichtet. Diagonale Wege stehen für eine Ausrichtung der beiden Zeichen, während horizontale und vertikale Wege Einfüge- und Löschoperationen repräsentieren (siehe [Böc07], S. 85). Existieren mehrere mögliche Pfade zurück zu dem Ursprung, so repräsentieren diese weitere mögliche Alignments der Sequenzen, welche (solange keine zusätzlichen Rahmenbedingungen gelten) ebenfalls als optimal anzusehen sind. Das vorgestellte LCS-Problem lässt sich im Übrigen auch als ein Spezialfall des *Global Alignment Problem* auffassen (siehe [Jon04], S. 178) und mittels des vorgestellten Algorithmus lösen.

Ein Beispiel wie der *Needleman-Wunsch-Algorithmus* ein optimales Alignment ermittelt, ist in der in Abbildung 5 abgebildeten Matrix *M* zu sehen. Dort soll ein optimales, globales Alignment der beiden Sequenzen *AATACG* und *ATGAC* gefunden werden. Als Ähnlichkeitsmatrix δ ist die in Abbildung 3 bereits vorgestellte Matrix zum Einsatz gekommen. Die Zahlen links oben in den einzelnen Feldern der Tabelle zeigen die jeweilige Güte *δ(w,x)* an. Die Werte der Felder sind nach der in Abbildung 5 vorgestellten Formel berechnet.

Der Wert von Feld M(*3,2*) berechnet sich beispielsweise aus

$$M(3,2) = \max\begin{cases} 8-6=2 \\ -12+-6=-18 \\ -6+8=2 \end{cases}$$

Der mittels Backtracking ermittelte Pfad, der zu einem optimalen Alignment führt, ist in Abbildung 5 gesondert hervorgehoben. Da er über das Feld M(*3,2)* führt, dessen Wert - wie gezeigt wurde - auf Basis zweier unterschiedlicher Formeln berechnet werden kann, existieren dementsprechend mehrere Möglichkeiten die Sequenzen optimal zueinander auszurichten. Abbildung 6 zeigt schließlich die beiden ermittelten optimalen globalen Alignments, welche in dem vorliegenden Beispiel auch einer LCS der beiden Sequenzen entsprechen.

	1	2	3	4	5	6	7
		A	**A**	**T**	**A**	**C**	**G**
1	*0* **0**	*-6* ← **-6**	*-6* -12	*-6* -18	*-6* -24	*-6* -30	*-6* -36
2	**A** *-6* -6	↖ *8* 8	↖ *8* ← **2**	*-1* -4	*8* -10	*-10* -16	*-3* -22
3	**T** *-6* -12	*-1* 2	*-1* 7	↖ *7* **9**	*-1* 3	*-2* -3	*-5* -9
4	**G** *-6* 18	*-3* -4	*-3* 1	↑ *-5* **3**	*-3* 6	*-6* 0	*10* 7
5	**A** *-6* -24	*8* -10	*8* 4	*-1* 0	↖ *8* **11**	*-10* 5	*-3* 1
6	**C** *-6* -30	*-10* -16	*-10* -2	*-2* 2	*-10* 5	↖ *9* **20**	*-5* ← 14

Abbildung 5: Matrix globales Alignment

```
Sequenz 1: -ATGAC-
Sequenz 2: AAT-ACG

Sequenz 1: A-TGAC-
Sequenz 2: AAT-ACG
```

Abbildung 6: Optimale Alignments (global)

2.3 Local Alignment Problem

Das vorherigen Kapitel vorgestellte *Global Alignment Problem* beschäftigt sich mit der Fragestellung, wie unter einer gegebenen Bewertungsfunktion eine optimale Ausrichtung zwischen zwei Sequenzen in ihrer Gänze zu berechnen ist. Diese Fragestellung ist allerdings nur sinnvoll, wenn zu erwarten ist, dass die betrachteten Sequenzen von Anfang an eine relativ hohe Ähnlichkeit, sowohl in ihrer Länge als auch in ihrem Inhalt, aufweisen. In biologischen Anwendungsgebieten ist dies allerdings oftmals nicht gegeben, beispielsweise wenn Proteinketten unterschiedlicher Herkunft zu vergleichen sind (siehe [Böc07], S. 90). Diese weisen hohe Ähnlichkeiten oftmals nur über Teilsequenzen auf, die jeweils biologisch einen ähnlichen Zweck erfüllen. Es kann in solchen Fällen vorkommen, dass ein Alignment zwischen zwei Teilsequenzen einen höheren Wert[8] aufweist, als wenn eine Ausrichtung der Sequenzen in ihrer Gesamtheit erfolgt. Es ist somit sofort verständlich, dass sich Ähnlichkeiten über Sequenzteile nicht mit Hilfe einer Lösung des Global Alignment Problem finden lassen. Somit ist für die gegebene Fragestellung ein neuer Ansatz zu finden. Im Weitern wird zunächst eine Definition des *Local Alignment Problem* gegeben und danach mit dem *Smith-Waterman-Algorithmus* eine Möglichkeit zur Beantwortung der Fragestellung aufgezeigt.

Jones (siehe [Jon04]) definiert auf S. 181 das Problem des Findens eines optimalen lokalen Alignments zwischen zwei Sequenzen wie folgt:

> *Local Alignment Problem:*
> *Find the best local alignment between two strings:*
>
> *Input: Strings v and w and a scoring matrix δ*
>
> *Output: Substrings of v and w whose global alignment, as defined by δ, is maximal among all global alignments of all substrings of v and w.*

Smith-Waterman-Algorithmus

Ein Verfahren zur Lösung der Fragestellung wurde 1981 von *Temple Smith* und *Michael Waterman* in ihrem Artikel *Identification of Common Molecular*

[8] *Für Beispiele wo ein optimales, globales Alignment keine Lösung für das lokale Alignment Problem aufzeigt siehe [Böc07], S .90 und [Jon04] S. 182, Figure 6.16.*

Subsequences [Smi81] vorgestellt. Der dort propagierte Algorithmus ist eine Variation des bereits eingehend beschriebenen *Needleman-Wunsch-Algorithmus*. Er nutzt dabei eine modifizierte Formel[9] (siehe Abbildung 7) zur Berechnung der Felder der Matrix *M*, sodass negative Feldwerte nicht mehr vorkommen können.

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + \delta(xi,yj) \\ M(i-1,j) - \delta(xi,-) \\ M(i,j-1) - \delta(-,yi) \\ 0 \end{cases}$$

Abbildung 7: Formel lokales Alignment

Auf diese Weise wird das Vorkommen von lokalen Alignments innerhalb der Matrix *M* positiv bewertet und somit sichtbar gemacht. Das Backtracking startet bei der Suche nach einem optimalen lokalen Alignment bei dem Feld mit dem höchsten Wert und wird solange durchgeführt, bis ein Eintrag mit dem Wert 0 erreicht ist.

In Abbildung 9 ist die Matrix *M* zu sehen, welche durch Einsatz des *Smith-Waterman-Algorithmus* bei der Suche nach einem optimalen, lokalen Alignment für die Sequenzen *GCTCAA* und *TACTCGG* entsteht. Als Ähnlichkeitsmatrix δ ist erneut die in Abbildung 3 gezeigte Matrix genutzt worden. Der Weg, der bei dem Backtracking vollzogen wird, ist analog zu dem Beispiel aus dem letzten Kapitel, auch hier gekennzeichnet. Das mit dem Algorithmus gefundene optimale lokale Alignment der beiden Sequenzen ist in Abbildung 8 dargestellt.

```
Sequenz 1: CTC
Sequenz 2: CTC
```

Abbildung 8: Optimales Alignment (lokal)

[9] *Alternativ ist es natürlich auch möglich eine gegebene Ähnlichkeitsmatrix δ direkt zu verändern und alle Vorkommen von negativen Werten auf 0 zu setzen.*

	1	2	3	4	5	6	7
		G	**C**	**T**	**C**	**A**	**A**
1	*0* 0	*-6* -6	*-6* -12	*-6* -18	*-6* -24	*-6* -30	*-6* -36
2	**T**	*-6* -6	*-5* 0	*-2* 0	*7* 0	*-2* 0	*-1* 0	*-1* 0
3	**A**	*-6* -12	*-3* **0**	*-10* 0	*-1* 0	*-10* 0	*8* 8	*8* 8
4	**C**	*-6* 18	*-6* 0	↖ *9* **9**	*-2* 3	*9* 9	*-10* 3	*-10* 2
5	**T**	*-6* -24	*-5* 0	*-2* 3	↖ *7* **16**	*-2* 10	*-1* 8	*-1* 2
6	**C**	*-6* -30	*-6* 0	*9* 9	*-2* 10	↖ *9* **25**	*-10* 19	*-10* 13
7	**G**	*-6* -24	*10* 0	*-6* 3	*-5* 4	*-6* 19	*-3* 22	*-3* 16
8	**G**	*-6* -30	*10* 0	*-6* 0	*-5* 0	*-6* 13	*-3* 16	*-3* 19

Abbildung 9: Matrix lokales Alignment

3. Fingerprintbasiertes Screening

Begriffsdefinition Fingerprint

Der Begriff *Fingerprint* wird in der Bioinformatik - und somit auch im Rahmen dieser Arbeit - von seiner Bedeutung her anders definiert als es normalerweise innerhalb der Informatik der Fall ist. Spricht man in der Informatik von einem Fingerprint, so ist üblicherweise ein Verfahren für das Abbilden eines beliebigen, oftmals großen Datums auf eine kürzere, in der Regel als Zeichenkette repräsentierte, Sequenz gemeint. Mittels dieser Sequenz soll das entsprechende Datum bei weiterer Verwendung innerhalb eines typischen Einsatzgebietes[10] für alle praktischen Zwecke eindeutig zu identifizieren sein.

Ein Fingerprint, der mit einem der beiden, im Folgenden näher vorgestellten, Verfahren gebildet wird, ist nicht für eine eindeutige Identifizierung eines Datums geeignet. Vielmehr beschreibt ein Fingerprint vor dem Hintergrund dieser Arbeit eine Menge an charakteristischen Merkmalen die ein Datum aufweist, bzw. welche charakteristischen Eigenschaften ihm fehlen. Sind demnach zwei an sich unterschiedliche Daten identisch in der Menge der abgebildeten Merkmale, so sind entsprechend auch ihre Fingerprints identisch. Folglich sind die hier vorgestellten Verfahren nicht in der Lage, eine eindeutige Identifikation eines Datums zu garantieren.

Ein Fingerprint ist im Folgenden immer als die Repräsentation einer Menge von charakteristischen Eigenschaften eines Datums zu verstehen. Welche Motivation hinter einem Einsatz von Fingerprints steckt, welche Arten von Fingerprints zu unterscheiden sind und auf welche Daten sie in der Bioinformatik angewendet werden wird in dem folgenden Kapitel näher erläutert.

---

[10] Der Begriff Fingerprint fällt häufig in Zusammenhang mit den Themen Kryptographie, Checksummenbildung und digitalen Signaturen. Es ist allerdings anzumerken, dass er auch in diesen Anwendungsgebieten teilweise ungenau und oftmals auch leicht unterschiedlich definiert ist.

3.1 Motivation für ein Screening

Eine oftmals im Kontext der Bioinformatik auftretende Fragestellung ist, ob eine chemische Struktur als Teilstruktur in einer weiteren zu vergleichenden chemischen Struktur enthalten ist. Der praktische Hintergrund der Fragestellung liegt darin begründet, dass chemische Strukturen, die sich in strukturellen Eigenschaften ähneln, auch in ihren weiteren Eigenschaften Ähnlichkeiten aufweisen. Dieses Konzept wird von *Gasteiger* (siehe [Gas03], S. 405) unter dem Motto *similar structure – similiar property principle* zusammengefasst.

Ein Anwendungsbeispiel für den Einsatz derartiger Ähnlichkeitsuntersuchungen ist die Entwicklung neuer Medikamente: Wenn einmal ein Wirkstoff für den Einsatz in einem Medikament identifiziert und somit auch seine chemische Struktur[11] bekannt ist, so ist es von Interesse, ob der Wirkstoff im Rahmen einer *Teil-von-Beziehung* auch in weiteren chemischen Strukturen zu finden ist. Existieren weitere Stoffe, die den gefundenen Wirkstoff enthalten, so ist es von Interesse zu untersuchen, ob sich durch eine Substitution der Wirkstoffe eine Verbesserung des Medikaments erzielen lässt. Als Beispiele von Verbesserungen sind an dieser Stelle exemplarisch das Erreichen eines höheren Wirkungsgrades oder eine Verringerung auftretender Nebenwirkungen genannt.

Wenn man dabei bedenkt, dass ein Molekül nichts anderes als eine Menge von Atomen darstellt, welche untereinander über Verbindungen in Beziehung stehen, so ist trivial ersichtlich, dass sich eine solche Struktur als ein Graph auffassen und dementsprechend repräsentieren (vgl.[Lea03], S. 2) lässt. Innerhalb eines solchen molekularen Graphen werden die jeweiligen Atome durch Knoten (engl. *nodes*), sowie die Verbindungen die zwischen Atomen existieren als Kanten (engl. *edges*) aufgefasst. Die im letzten Abschnitt angesprochene Problemstellung des Suchens und Findens der Struktur eines Moleküls in einer weiteren chemischen Struktur ist demnach identisch mir der Problemstellung der *Subgraph Isomorphie*[12] aus der Graphtheorie. Bei diesem Entscheidungsproblem gilt es herauszufinden, ob bei zwei gegeben Graphen *G1* und *G2* der Graph *G1* isomorph zu einem Subgraphen von *G2* ist. Die Komplexität dieses Entscheidungsproblems liegt nachweislich in dem Bereich der *NP-Vollständigkeit* (siehe [Lea03], S. 12). Die Laufzeit für die Berechnung einer Lösung steigt damit exponentiell mit der Größe des Problems, in diesem Fall repräsentiert durch die Anzahl der Knoten der Graphen.

[11] *Der Begriff* chemische Struktur *bezeichnet allgemein den Aufbau eines Stoffes auf molekularer Ebene und wird im Rahmen dieser Arbeit synonym mit dem Begriff Molekül verwendet.*

[12] Oft wird in der Bioinformatik auch der Begriff *Substruktur Suche* (engl. *Substructure Search*) synonym verwendet, da sich die Begriffswelt der Bioinformatik näher an den biologisch/chemischen Grundlagen orientiert, als an einer abstrakten Darstellung der Strukturen in Form eines Graphen.

Für die vorliegende Problemstellung des Findens einer Subgraph Isomorphie zwischen zwei chemischen Strukturen lässt sich mittels geeigneter Annahmen die allgemeine Problematik für den vorliegenden Spezialfall einschränken. So wurden in der Bioinformatik (siehe [Day08]) effizientere Algorithmen für die vorliegende Ausprägung des Problems entwickelt, welche eine Antwort auf die Fragestellung typischerweise mit Laufzeitkomplexitäten im Bereich von $O(N^2)$ bis $O(N^3)$ ermitteln. Laufzeitenkomplexitäten dieser Größenordnung sind, obwohl sie eine deutliche Verbesserung gegenüber dem exponentiellen Fall darstellen, aus algorithmischer Sicht immer noch äußerst langsam. Besonders wenn statt einer einzelnen chemischen Struktur eine Menge an Molekülen, beispielsweise persistent in einer chemischen Datenbank hinterlegt, auf Subgraph Isomorphie zu einem Mustermolekül *P* zu testen ist, führen auch die verbesserten Algorithmen in der Praxis nicht zu zufriedenstellenden Laufzeiten (vgl. [Lea03], S. 9). Dementsprechend sind für viele Anwendungsfelder weitere Ansätze die auf eine Verbesserung der Laufzeit zielen von großem Interesse.

Ein erster einfacher Ansatz (vgl. [Day08]) stützt sich auf die Idee die Fragestellung des Problems einfach zu spiegeln. Es lässt sich zwar nicht effizient entscheiden, ob eine chemische Struktur *P* in einem Molekül *M* vorkommt, sehr wohl kann aber in linearer Zeit entschieden werden, ob das Muster *P* überhaupt in *M* enthalten sein kann. Dabei gilt allerdings die Einschränkung, dass nur das Vorkommen einer Subgraph Isomorphie mit Sicherheit ausgeschlossen werden kann. Sollte indes bei zwei Molekülen festgestellt werden, dass eine Subgraph Isomorphie vorliegen kann, so ist in weiteren Schritten noch zu überprüfen, ob das Muster *P* auch tatsächlich in dem Molekül *M* enthalten ist.

Um besser verständlich zu machen wie ein solches Verfahren für molekulare Strukturen genutzt werden kann, folgendes Beispiel: Für eine gegebene Moleküldatenbank werden in einem ersten Schritt zu jeder hinterlegten chemischen Struktur die jeweils vorkommenden Atome errechnet und gespeichert. Dies stellt zwar einen einmaligen Aufwand dar, der allerdings vertretbar ist. Einmal gewonnene Informationen können bei jeder folgenden Anfrage erneut genutzt werden und der Anfangsaufwand amortisiert sich somit bei steigender Anzahl der Anfragen. Wenn nun ein Mustermolekül *P* gegen die chemischen Strukturen in der Datenbank zu vergleichen ist, errechnet man auch dessen Atome und vergleicht das Ergebnis mit den zuvor errechneten Ergebnissen der chemischen Strukturen aus der Datenbank.

Enthält ein Molekül aus der Datenbank nicht alle in dem Muster vorkommenden Atome, so kann es trivialerweise kein Kandidat für eine Subgraph Isomorphie sein und kann demnach von einer weiteren Betrachtung mittels aufwendiger Algorithmen ausgeklammert werden. Ob ein Molekül, welches alle Atome des Musters enthält, gleichsam auch das Muster als Substruktur aufweist, kann wie bereits beschrieben mit diesem Ansatz nicht eindeutig entschieden werden. Sehr wohl verhindert dieser

Ansatz allerdings, dass Moleküle, die keine Aussicht auf eine Subgraph Isomorphie besitzen, mittels der ineffizienten Algorithmen aufwendig zu untersuchen sind.

Einen solchen Ansatz für die Filterung einer Menge von Kandidaten wird allgemein als *Screening* bezeichnet. Ziel eines Screenings ist eine möglichst hohe Menge an Kandidaten auszuschließen (vgl. [Lea03], S. 9). Das hier vorgestellte Verfahren auf Basis der in den chemischen Strukturen vorkommenden Atome ist dabei als ein erster simpler Ansatz anzusehen. Die im Folgenden vorgestellten Fingerprintverfahren greifen allerdings auf das in diesem Abschnitt beschriebene Prinzip zurück und bieten gleichzeitig zusätzliche und differenziertere Möglichkeiten, um die Filterung einer geeigneten Kandidatenmenge durchzuführen.

3.2 Strukturelle Fingerprints

Einen ersten Ansatz für die Durchführung eines verbesserten Screenings stellen *strukturelle Fingerprints* (engl. *Structural Keys*) dar. Strukturelle Fingerprints sind eine abstrakte Repräsentation einer Menge von Merkmalen (es werden auch oft synonym die Begriffe *Features*, *Eigenschaften* oder *Muster* verwendet) einer chemischen Struktur. Die Abbildung dieser Merkmale erfolgt in der Regel mittels einer Bitfolge (siehe [Lea03], S. 9). Eine solche Repräsentation hat den Vorteil, dass sich binäre Datenstrukturen mit Computern schnell und effizient verarbeiten lassen. Jede Bitposition in der Folge steht dabei für ein Merkmal und der jeweilige Wert zeigt an, ob das Feature in dem betrachteten Molekül präsent ist (durch den Wert *1* ausgedrückt) oder fehlt (die entsprechende Bitposition besitzt den Wert *0*).

Eine Liste von Merkmalen, die in einem Fingerprint abzubildende Features definiert, wird als Musterkatalog bezeichnet. Es ist sofort ersichtlich, dass die Länge eines Fingerprints bei diesem Ansatz von der Anzahl der in ihm beschriebenen Merkmale, respektive der Länge des Musterkatalogs, abhängig ist.

Die Merkmale, die sich in einem strukturellen Fingerprint abbilden lassen, sind willkürlich wählbar. Einige typische Merkmale und Eigenschaften nach [Gas03] sind beispielsweise:

- Vorkommen und Anzahl eines Atoms (vgl. Beispiel Einleitung)
- Vorkommen und Anzahl einer chemischen Struktur
- Vorkommen einer oder mehrerer *Funktioneller Gruppen*[13]
- Topologische Deskriptoren der chemischen Struktur
- Geometrische Deskriptoren der chemischen Struktur
- Oberflächeneigenschaften der chemischen Struktur

Es ist festzuhalten, dass die Anzahl an Merkmalen, die in einem Fingerprint repräsentiert sein können, im Prinzip unendlich ist. Man denke dabei an die Abbildung des Vorkommens einer beliebigen chemischer Struktur als Merkmal eines Fingerprints. Erweitert man das Molekül um ein einzelnes Atom, so erhält man eine neue Struktur, welche sich wieder als ein eigenes Merkmal auffassen und abbilden lässt. Dieser Vorgang ist, zumindest theoretisch, beliebig oft wiederholbar.

[13] *Eine funktionelle Gruppe ist eine Atomgruppe in einer organischen Verbindung, welche das Reaktions- und das Stoffverhalten der Verbindung maßgeblich bestimmt. Kommt eine funktionelle Gruppe in mehreren Verbindungen vor, so ist davon auszugehen, dass die Verbindungen sich in den durch die Gruppe bestimmten Eigenschaften gleichen.*

Der Ablauf der Erstellung eines strukturellen Fingerprints ist in Abbildung 10 in Anlehnung an *Kohlbacher* [Koh06] zu sehen. Das abgebildete Molekül wird auf ein Vorkommen der einzelnen Merkmale - hier charakteristische chemische Strukturen - untersucht. Wenn dabei das Vorkommen einer Verbindung in der chemischen Struktur bestätigt ist, wird die entsprechende Bitstelle in dem Fingerprint gesetzt.

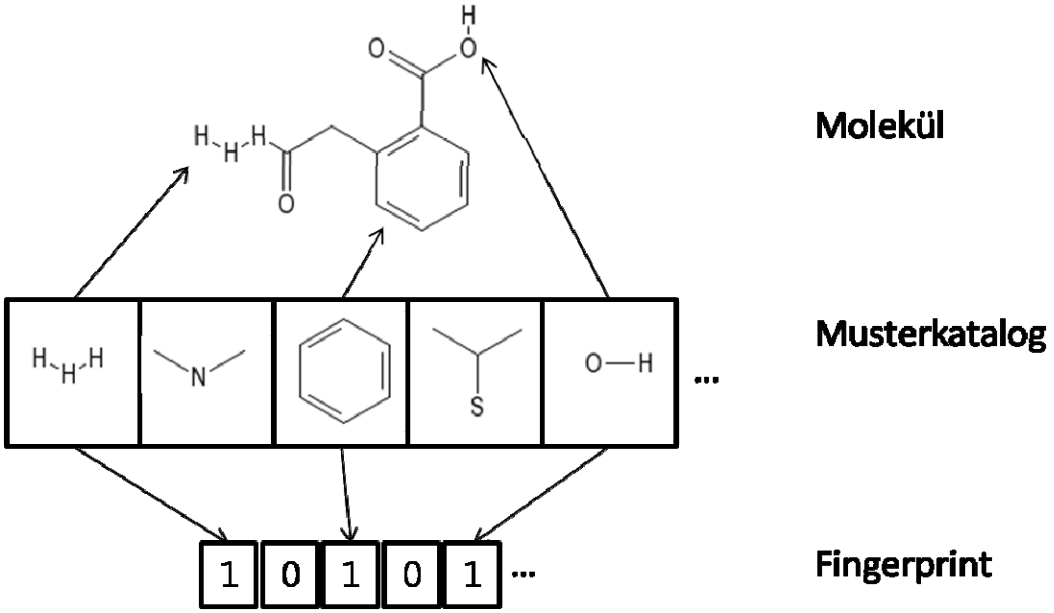


Abbildung 10: Schematischer Ablauf der Erstellung eines strukturellen Fingerprints

Das Screening erfolgt dabei analog zu dem in der Einleitung beschriebenen Ablauf. Wenn eine Menge von chemischen Strukturen auf Basis ihres strukturellen Fingerprints nach Kandidaten für eine Übereinstimmung mit einem Mustermolekül *P* gescreent werden soll, so wird ein Fingerprint für das zu suchende Muster erstellt. Dies geschieht auf Basis des gleichen Musterkatalogs, der auch zuvor bei der Erstellung der Fingerprints der zu vergleichenden Moleküle zum Einsatz kam. In einem zweiten Schritt wird dann der erstellte Fingerprint des Musters gegen alle Fingerprints der weiteren Moleküle verglichen. Wenn eine chemische Struktur nicht alle repräsentierten Merkmale des gesuchten Musters aufweist, wird sie nicht in die Kandidatenmenge aufgenommen. Die Moleküle, die alle Merkmale erfüllen, werden dann mittels weiterer Verfahren genauer untersucht. Neben der bereits angesprochenen Untersuchung mittels komplexer Algorithmen geschieht eine solche Prüfung oft auch mittels Laborexperiment. Dies ist gerade bei der in der Einleitung angesprochenen Zielsetzung, einen besseren Wirkstoff für ein Medikament zu

finden, von Interesse. Es lässt sich mittels einer computergestützten Simulation oftmals nicht endgültig entscheiden, ob der Einsatz eines gefundenen Kandidaten einen Vorteil gegenüber dem schon bekannten Wirkstoff erzielt.

Wenn ein Musterkatalog dabei nur aus sehr speziellen Merkmalen zusammengesetzt ist, so kann dies zu spärlich besetzten Fingerprints führen. Betrachtete chemische Strukturen werden in der Regel nur wenige der speziellen Merkmale erfüllen. Spärlich besetzte Fingerprints sind als ein Resultat ihrer geringen Informationsdichte ineffizient und sollten nach Möglichkeit schon bei der Erstellung eines Musterkataloges vermieden werden. Lassen sich aber in einem gegebenen Anwendungsfall spärlich besetzte Fingerprints nicht vermeiden, kann zu einer Einsparung von Speicherplatz und einer Erhöhung der Informationsdichte des Fingerprints eine *Faltung* (engl. *folding*) (siehe [Day08]) durchgeführt werden. Dabei wird jeder Fingerprint in seiner Mitte geteilt und die beiden resultierenden Hälften mittels einer logischen Oder-Funktion verknüpft. Ein Fingerprint der durch eine solche Verknüpfungsoperation entsteht beschreibt natürlich nicht mehr mit jeder Bitstelle eindeutig ein Merkmal. Jede gesetzte Position gibt also im Weiteren nur noch den Hinweis auf das Vorkommen einer Eigenschaft. Bei Bedarf kann eine solche Faltung wiederholt durchgeführt werden.

Strukturelle Fingerprints besitzen einige Nachteile. Zum einen erlauben sie in dieser ursprünglichen Form nur harte Kriterien für ein Screening: Erfüllt eine chemische Struktur auch nur eines der gesuchten Merkmale nicht, wird sie aussortiert. Dies geschieht auch, wenn die chemische Struktur trotz der Abweichung eine sehr hohe Ähnlichkeit - und dementsprechend wahrscheinlich auch ähnliche Eigenschaften - zu dem gesuchten Muster aufweist. Ein Ansatz der diesen Kritikpunkt anspricht wird im Laufe der Arbeit noch genauer vorgestellt. So gibt es in der Praxis bewährte Ähnlichkeits- und Differenzmaße (siehe *Kapitel 3.4 Ähnlichkeitsberechnung zwischen Fingerprints*), die es ermöglichen einen Ähnlichkeitswert zwischen zwei chemischen Strukturen auf Basis ihrer Fingerprints zu ermitteln. Durch einen solchen Ansatz lassen sich bei der Suche nach einem Mustermolekül die chemischen Strukturen, welche eine hohe Ähnlichkeit in ihren Eigenschaften aufweisen, finden. Ebenso ist es - wie später gezeigt wird – möglich, einen Schwellenwert zu definieren den ein Kandidat als Mindestähnlichkeit zu einem gesuchten Molekül aufweisen muss, um als grundsätzlich ähnlich angesehen zu werden.

Der größte Kritikpunkt an strukturellen Fingerprints liegt allerdings in der Nutzung eines Musterkatalogs, da er eine äußert unflexible Methode zur Festlegung der Merkmale darstellt. Die abzubildenden Eigenschaften sind vor dem ersten Einsatz festzulegen und im Weiteren nicht mehr problemlos an neue Gegebenheiten anpassbar. Der resultierende Musterkatalog ist entsprechend starr in den repräsentierten Merkmalen. Weiterhin beeinflusst die Wahl der abzubildenden Merkmale sowohl die Geschwindigkeit bei einer Suche nach ähnlichen chemischen

Strukturen, als auch die Qualität der ermittelten Kandidatenmenge (vgl. [Bra00], S. 228). Insbesondere Fingerprints, in denen nur wenige Merkmale präsent sind, verhindern mitunter ein effektives Screening der Kandidatenmenge.

Die Problematik der Auswahl sinnvoller Merkmale für einen Musterkatalog verstärkt sich, wenn eine einzelne molekulare Datenbank in verschiedenen Anwendungsgebieten zum Einsatz kommt. Es ist plausibel, dass nicht jedes Merkmal einer chemischen Struktur für jedes denkbare Einsatzgebiet von Interesse ist. So lässt sich ein Fingerprint der in einem pharmazeutischen Kontext zum Einsatz kommt, nicht automatisch für eine Anwendung in der Petrochemie (vgl. [Day08]) nutzen. Es ist in diesen Fällen abzuwägen, ob entweder ein einziger Musterkatalog für alle gegeben Anwendungsfälle zu definieren ist oder ob es sinnvoller erscheint für jeden Anwendungsfall einen eigenen Musterkatalog zu definieren.

3.3 Hashed Fingerprints

Hashed Fingerprints sind eine Weiterentwicklung der strukturellen Fingerprints, welche versucht einige der im letzten Kapitel angeführten Kritikpunkte zu eliminieren. Bei der Erstellung eines Hashed Fingerprints wird auf den Einsatz eines Musterkataloges verzichtet und stattdessen errechnet sich der jeweilige Fingerprint direkt aus der Struktur des betrachteten Moleküls.

Um den Hashed Fingerprint einer chemischen Struktur zu erstellen, werden alle in dem Molekül vorhandenen Teilstrukturen (im Folgenden auch als *Fragmente* bezeichnet) bis zu einer bestimmten Größe[14] betrachtet. Da es nur eine endliche Anzahl unterschiedlicher Atome gibt und nur Fragmente bis zu einer festgelegten Größe untersucht werden, ergibt sich daraus, dass die Menge aller denkbaren Fragmente endlich und somit auch aufzählbar ist. Aus diesem Grund kann letztendlich jedem Fragment eine eindeutige *ID* zugewiesen werden (vgl. [Koh06]). Eine solche Fragment-ID wird im Folgenden bei der Erstellung eines Hashed Fingerprints als Startparameter für eine Hashfunktion *f(ID)* genutzt. Typische Hashfunktionen bilden dabei das jeweilige Fragment auf vier bis fünf Bitpositionen in einem Fingerprint ab.

Bei diesem Ansatz ersetzen die jeweiligen Fragmente somit die in einem strukturellen Fingerprint genutzten Merkmale als Grundlage der charakteristischen Repräsentation einer chemischen Struktur.

Sind auf diese Weise für alle in einem vorliegenden Molekül gefundenen Fragmente die Fingerprints erstellt, werden diese in einem letzten Schritt mittels einer logischen OR-Verknüpfung zu einem einzigen Fingerprint aggregiert. Abbildung 11 (in Anlehnung an *Kohlbacher* [Koh06]) zeigt dabei schematisch den Ablauf der Erstellung eines Hashed Fingerprints. Es werden dabei nur Fragmente der Länge drei berücksichtigt.

[14] *In der Praxis werden häufig alle Fragmente mit einer Länge zwischen 2 und 7 Verbindungen untersucht (siehe [Day08]).*

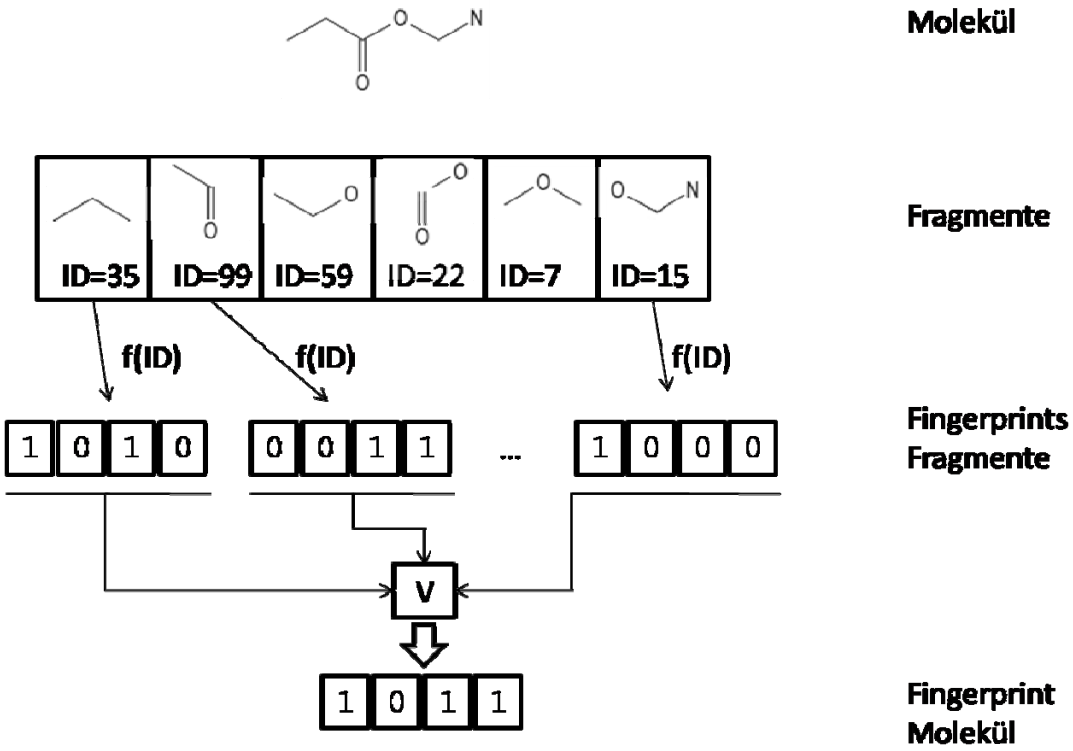


Abbildung 11: Schematischer Ablauf der Erstellung eines Hashed Fingerprints

Hashed Fingerprints eliminieren die Notwendigkeit eines Musterkataloges oder sonstiger zusätzlicher Hilfsmittel für die Berechnung des Fingerprints einer chemischen Struktur. Allerdings ist anzumerken, dass sie einen Unterschied in ihrer Aussagekraft gegenüber strukturellen Fingerprints aufweisen. Durch eine Bitstelle in einem strukturellen Fingerprint ist eindeutig[15] erkennbar ob das jeweilige Merkmal in dem Molekül vorhanden ist oder nicht (vgl. [Bra00], S. 229). Wenn in einem Hashed Fingerprint allerdings diejenigen Bitstellen, die ein Fragment repräsentieren, gesetzt sind, so ist dies nur ein Hinweis auf ein mögliches Vorkommen. Durch Überschneidungen bei der Abbildung verschiedener Fragmente kann in ungünstigen Fällen eine Kombination auftreten, die es so aussehen lässt, als ob ein Fragment in der chemischen Struktur vorkommt das nicht vorhanden ist. Ein Beispiel dafür ist auch in Abbildung 11 zu entdecken. Die Oder-Verknüpfung der Fingerprints über die Fragmente *ID=99* und *ID=15* ergibt einen Hinweis auf das Vorhandensein des Fragments *ID=35*, unabhängig davon ob es tatsächlich in dem Molekül enthalten ist.

[15] *Solange keine Faltung vorgenommen wurde.*

Aus diesem Grund ist bei einem Einsatz von Hashed Fingerprints, ähnlich dem zu Beginn des Kapitels aufgeführten Beispiels, im Weiteren noch genau zu überprüfen, ob eine gesuchte chemische Struktur tatsächlich in einem gefundenen Kandidaten enthalten ist. Abgesehen von dem möglichen Auftreten eines *false-positive* gilt natürlich auch für einen Hashed Fingerprint, dass ein Fragment mit Sicherheit nicht in einer chemischen Struktur enthalten ist, wenn nicht alle repräsentativen Bitstellen den Wert *1* aufweisen. Weiterhin ändert dies auch nichts an der grundsätzlichen Annahme, dass ähnliche chemische Strukturen auch durch ähnliche Fingerprints repräsentiert sind.

Hashed Fingerprints sind für alle Datenbanken, in denen chemische Strukturen gespeichert werden, sofern das zugrundeliegende Datenformat bekannt ist, ohne weitere Konfigurationen einsetzbar. Durch den Einsatz der Hashfunktion werden die Bitstellen im Fingerprint wesentlich effizienter ausgenutzt als bei strukturellen Fingerprints. Bei einer Betrachtung von Pfaden einer Länge zwischen *2* und *7* Verbindungen gibt es insgesamt etwa 10^5 mögliche Fragmente (vgl. [Koh06]) die in einem Hashed Fingerprint repräsentiert werden können. Somit übersteigt die Anzahl der in einem Hashed Fingerprint darstellbaren Merkmale deutlich die Anzahl der Merkmale, die in strukturellen Fingerprints noch praktikabel abzubilden sind. Zu spärlich besetzten Fingerprints kann es auch nur kommen, wenn chemische Strukturen, die aus wenigen Fragmenten bestehen, untersucht werden. Sollte dies allerdings in einem gegebenen Anwendungskontext verstärkt auftreten, so besteht analog zu strukturellen Fingerprints die Möglichkeit eine Faltung vorzunehmen. Komplexe chemische Strukturen (vgl. [Day08]) hingegen werden mittels des vorgestellten Verfahrens sehr akkurat durch ihren Fingerprint beschrieben, da sie in der Regel distinkte, sich überlappende Fragmente enthalten.

Der größte zu nennende Nachteil von Hashed Fingerprints liegt in der Tatsache, dass neben dem Vorhandensein der Fragmente keine zusätzlichen Informationen über die chemische Struktur hinterlegt werden können. Moleküleigenschaften, beispielsweise 3D-Informationen oder die Bindungseigenschaften der Struktur, sind, sofern sie für die jeweilige Fragestellung von Interesse sind, in weiteren Schritten zu ermitteln. [16]

Abschließend bleibt festzuhalten, dass Hashed Fingerprints für viele Anwendungsgebiete eine Reihe von Vorteilen gegenüber strukturellen Fingerprints aufweisen. Dies trifft insbesondere auf Anwendungsgebiete zu, bei denen sich typische Fragestellungen auf das Finden von chemischen Strukturen beschränken. Sollten für eine effektive Suche nach Kandidaten allerdings weitere Eigenschaften

[16] *Eine Ausnahme dabei ist natürlich, wenn sich aus den betrachteten Fragmenten Rückschlüsse auf die gesuchte Information ziehen lassen. So zum Beispiel wenn ein Fragment Repräsentant einer funktionellen Gruppe ist. Aber auch in einem solchen Fall ist erst eine weitere Überprüfung nötig, bevor ein false-positiv ausgeschlossen und ein Vorkommen der funktionellen Gruppe mit Sicherheit bestätigt werden kann.*

zu berücksichtigen sein, so ist im Einzelfall zu prüfen, ob der Einsatz von Hashed Fingerprints sinnvoll ist.

3.4 Ähnlichkeitsberechnung zwischen Fingerprints

In dem folgenden Kapitel werden Möglichkeiten erläutert, auf welcher Basis die Bestimmung eines Ähnlichkeitswerts zwischen zwei Fingerprints erfolgen kann. Dabei ist es nicht weiter von Interesse, mit Hilfe welches der beiden vorgestellten Verfahren die Fingerprints erstellt wurden. Allerdings wird von einem Vorkommen der im letzten Kapitel angesprochenen *false-positives* für Hashed Fingerprints abstrahiert, da sie eine verfahrensimplizite Problematik darstellen. Es ist klar, dass ein etwaiges Auftreten auch nach einer fingerprintbasierten Ähnlichkeitsbestimmung noch durch weitere Untersuchungen auszuschließen ist. Bevor nun einige der in der Praxis gängigen Ähnlichkeitsmaße vorgestellt und näher erläutert werden, werden als Einleitung zwei einfache Methoden zur Bestimmung der Ähnlichkeit zwischen Fingerprints dargestellt.

Einleitung

Ein erster Ansatz zu der Bestimmung einer Ähnlichkeit zweier Fingerprints ist die Prüfung auf Identität. Dabei sind zwei Fingerprints trivialerweise als identisch anzusehen, wenn sie in allen repräsentierten Merkmalen - folglich auch in allen Bitpositionen - übereinstimmen. Überprüfen lässt sich eine Identitätsbeziehung durch eine logische *XOR*-Verknüpfung der beiden Fingerprints. Wenn in der aus der Verknüpfung resultierenden Bitfolge alle Stellen auf *0* gesetzt sind, so liegt eine Identität vor.

Das in *Kapitel 3.1 Motivation* genutzte Beispiel für ein Screening einer molekularen Datenbank, in dem eine chemische Struktur als Kandidat anzusehen ist, wenn sie mindestens alle Eigenschaften aufweist, die auch in dem zu suchenden Muster enthalten sind, ist Ausdruck einer weiteren einfachen Ähnlichkeitsbeziehung. In diesem Fall wird untersucht, ob eine *Teil-von*-Beziehung zwischen den Merkmalen des Musters und den Merkmalen eines Kandidaten besteht. Dies lässt sich durch eine logische *AND*-Verknüpfung der Fingerprints realisieren. Sollte der aus der Verknüpfung resultierende Fingerprint identisch zu

dem Fingerprint des Musters sein, so ist daraus zu schließen, dass alle Merkmale des Musters auch in dem jeweiligen Kandidaten vorkommen.

Diese beiden einfachen Arten einer Ähnlichkeitsbestimmung reichen allerdings in einigen Anwendungskontexten nicht aus. Vielmehr ist es nötig einen Ähnlichkeitswert zu ermitteln, der es im Weiteren ermöglicht, einzelne Paarungen von Fingerprints zu bewerten und verschiedene Paarungen zueinander in Relation zu setzen. Dass sich eine solche Berechnung natürlich nur auf die in Fingerprints gesetzten und nicht gesetzten Bitpositionen stützen kann, ist sofort ersichtlich. Auf welche Weise gesetzte, bzw. nicht gesetzte Bitpositionen bei einer Berechnung des Ähnlichkeitswerts berücksichtigt werden, ist allerdings mitunter sehr unterschiedlich.

3.4.1 Ähnlichkeits- und Distanzmaße

In dem nächsten Abschnitt erfolgt eine Übersicht ausgewählter Vergleichsmaße zur Berechnung eines Ähnlichkeitswerts zwischen zwei binären Fingerprints. Es werden ferner die unterschiedlichen Faktoren, die in die jeweilige Berechnung einfließen, näher erläutert. Die vorgestellten Maße repräsentieren dabei nur eine kleine Auswahl der in der Literatur vorgeschlagenen Möglichkeiten zur Bestimmung der Ähnlichkeit. Für Definitionen weiterer Maße sei an dieser Stelle auf die Aufzählung in[Gas03] auf S. 305ff verwiesen.

Um eine klare Begriffswelt für die Darstellung der im Folgenden beschriebenen Ähnlichkeits- und Distanzmaße zu schaffen, liegt diesem Abschnitt folgende Definition zu Grunde:

Gegeben sind zwei Fingerprints A und B. Beide Fingerprints sind in ihrer Länge identisch und eine Bitposition repräsentiert in beiden Fingerprints das gleiche Merkmal. Weiter sei das Vorkommen eines Merkmals angezeigt, indem die entsprechende Bitposition den Wert 1 aufweist. Im Folgenden sei:

a: die Anzahl der Merkmale, die in A vorkommen, aber in B fehlen.
b: die Anzahl der Merkmale, die in B vorkommen, aber in A fehlen.
c: die Anzahl der Merkmale, die in beiden Fingerprints vorkommen.
d: die Anzahl der Merkmale, die in beiden Fingerprints fehlen.
N: die Anzahl der abgebildeten Merkmale (a+b+c+d).

Einfacher Ähnlichkeits-Koeffizient

Der einfache Ähnlichkeits-Koeffizient setzt die in beiden Fingerprints identisch abgebildeten Merkmale in ein Verhältnis zu der Summe aller abgebildeten Merkmale. Dabei werden gesetzte und nicht gesetzte Merkmale als gleichwertig angesehen.

Einfacher Ähnlichkeits-Koeffizient

$$S_{AB} = \frac{c+d}{\mathrm{N}}, \text{ mit } S_{AB}[0 \dots 1]$$

Es ist bei einem Einsatz des vorgestellten Koeffizienten zu beachten, dass Übereinstimmungen zweier Fingerprints in fehlenden Attributen nicht für alle Fragestellungen eine Relevanz besitzen[17]. Weiterhin ist die Aussagekraft des errechneten Wertes abhängig davon, wie Aussagekräftig die abgebildeten Merkmale sind. Kommt es aus den in *Kapitel 3.2 Strukturelle Fingerprints* angeführten Gründen zu spärlich besetzten Fingerprints, so verzerrt dies mitunter das Ergebnis der Ähnlichkeitsberechnung. Insbesondere kleine Moleküle scheinen sich - da sie in vielen fehlenden Merkmalen übereinstimmen - ähnlicher zu sein, als dies tatsächlich der Fall ist.

Hamming-Distanz

Die schon in *Kapitel 2.1 Sequence Alignment* vorgestellte Hamming-Distanz ist definiert als die Anzahl an Stellen, in denen sich zwei Sequenzen unterscheiden. Um die Hamming-Distanz zwischen Fingerprints zu berechnen wird - ebenso wie bei einer Prüfung auf Identität - die logische *XOR*-Verknüpfung genutzt. Die Anzahl der in der resultierenden Bitfolge gesetzten Stellen entspricht dabei der Distanz der beiden Fingerprints nach Hamming. Ist der Wert 0, so sind die beiden Fingerprints in den repräsentierten Merkmalen identisch. Die Hamming-Distanz ist ein Vertreter der metrischen Distanzmaße (vgl. [Lea03] S. 103).

Hamming Distanz

$$D_{AB} = a + b, \text{ mit } D_{AB}[0 \dots N]$$

[17] *Ein in der Literatur gern genommenes humoristisches Beispiel für diese Feststellung aus dem Bereich der Evolutionsbiologie ist die Anmerkung, dass sich Menschen und Fische nicht automatisch ähnlicher sind, nur weil beide Spezies das Merkmal „Flügel" nicht aufweisen.*

Tanimotokoeffizient

Der Tanimotokoeffizient ist das in der Praxis meist genutzte Ähnlichkeitsmaß für den Vergleich binärer Fingerprints. Er berücksichtigt bei seiner Berechnung einzig Merkmale, welche in den Fingerprints vorkommen. Das Komplement des Tanimotokoeffizienten erfüllt, zumindest für dichotome Variablen, alle Eigenschaften einer Metrik (siehe [Lea03], S.101).

Tanimotokoeffizient

$$S_{AB} = \frac{c}{\mathrm{a}+b+c}, \text{ mit } S_{AB}[0 \ldots 1]$$

Tversky-Index

Der Tversky-Index (siehe [Lea03], S. 103) repräsentiert einen asymmetrischen Index. Für die Berechnung werden zwei Gewichte α und β eingeführt, die jeweils angeben wie stark das Auftreten bzw. das Fehlen eines Merkmals in der Berechnung der Ähnlichkeit zu berücksichtigen ist. Für eine errechnete Ähnlichkeit zwischen zwei Fingerprints kann demnach der Fall $S_{AB} \neq S_{BA}$ auftreten. Somit erfüllt der Tversky-Index nicht alle Eigenschaften eines metrischen Ähnlichkeitsmaßes.

Tversky-Index

$$S_{AB} = \frac{c}{\alpha(a-c) + \beta(b-c) + c}$$

Für $\alpha = 1$ und $\beta = 0$ reduziert sich der Bruch zu $\frac{c}{a}$. In diesem Fall repräsentiert der Tversky-Index den Anteil an Attributen von Fingerprint A, die auch in Fingerprint B vorkommen. Für den Spezialfall $\alpha = 1$ und $\beta = 1$ liefert der Tversky-Index das gleiche Ergebnis wie der Tanimotokoeffizient.

3.4.2 Ähnlichkeitsabfragen und Nachbarschaft

Ist ein metrischer Ansatz für die Berechnung eines Ähnlichkeitswertes festgelegt, so gibt es im Weiteren zwei Möglichkeiten zur Bestimmung einer aussichtsreichen Kandidatenmenge. Eine solche Bestimmung kann auf Basis

- der Entfernung der Elemente
- durch Festlegung eines Schwellenwerts erfolgen.

Bei der Suche über die Entfernung wird keine Vorgabe gemacht, wie groß die Distanz eines Elements zu einem potentiellen Kandidaten maximal sein darf. Stattdessen lautet die Fragestellung schlicht *„Was sind die x ähnlichsten Elemente?“*, wobei *x* die gewünschte Anzahl an Kandidaten repräsentiert. Wenn es für ein Element nur Kandidaten gibt, die eine große Distanz aufweisen, so werden sie trotzdem in die Kandidatenmenge aufgenommen.

Anders verhält es sich bei der Bestimmung einer Kandidatenmenge mit Hilfe eines Schwellenwertes. In diesem Fall ist die Anzahl der in der Menge vorkommenden Elemente nicht beschränkt, sondern kann im Extremfall aus allen betrachteten Elementen bestehen. Die Entscheidung, ob ein Element aufgenommen wird, basiert dabei einzig auf dem berechneten Abstand zu dem gesuchten Element. Sobald dieser den festgelegten Schwellenwert überschreitet, wird das Element als potentieller Kandidat in die Menge aufgenommen.

Für den Einsatz innerhalb eines Fingerprintverfahrens erweist sich ein Ansatz auf Basis eines Schwellenwertes als sinnvoller. Hier kann der Fall, dass ein Kandidat trotz großer Distanz zu dem gesuchten Element einer genauen Untersuchung unterzogen wird, durch das Festlegen eines geeigneten Schwellenwertes vermieden werden. Weiterhin ist es bei einem Ansatz über die Entfernung nicht nur nötig die einzelnen Distanzwerte der Elemente zu errechnen, sondern die berechneten Werte sind später zusätzlich zu sortieren, damit eine Bestimmung der ähnlichsten Elemente in der vorher festgelegten Anzahl erfolgen kann.

4. Implementierung

Zu Beginn des Kapitels wird ein Algorithmus zur Differenzanalyse von Dokumenten vorgestellt. Dabei wird aufgezeigt, an welcher Stelle der Einsatz eines fingerprintbasierten Screenings eine Optimierung der Laufzeit ermöglichen kann. Danach werden Unterschiede in den Rahmenbedingungen zwischen einem Screening von molekularen Datenbanken und einem Screening bei der Differenzanalyse von Dokumenten aufgezeigt. Auf Basis der Implementierung des vorgestellten Differenzalgorithmus (*SiDiff*) ist im Weiteren die Implementierung eines fingerprintbasierten Screenings skizziert. Das Kapitel schließt mit einer Darstellung von Möglichkeiten, wie verschiedene Ausprägungen von Eigenschaften in einem Fingerprint abgebildet werden können.

4.1 Ein Algorithmus zur Berechnung von Dokumentdifferenzen (SiDiff)

In der Arbeit *Ein XMI-basiertes Differenzwerkzeug für UML-Diagramme* [Weh04] wird von *Wehren* der Prototyp eines Algorithmus vorgestellt, der die Differenzen zwischen den Elementen aus zwei technischen Dokumenten ermittelt. Ziel der Differenzierung ist dabei die Erzeugung eines vereinheitlichen Dokumentes, welches alle Elemente aus den zu vergleichenden Dokumenten umfasst und in dem auftretende Differenzen durch Annotationen nachvollziehbar (siehe [Weh04], S. 2) sind.

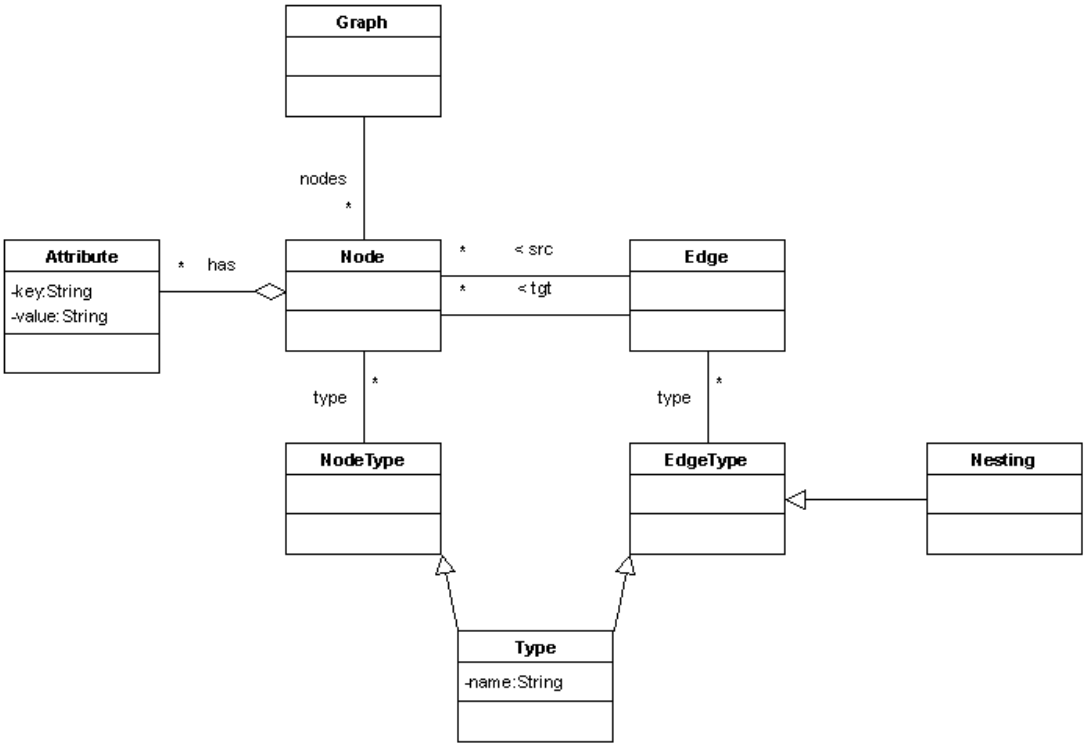


Abbildung 12: Internes Datenmodell für die Differenzberechnung

Der eigentliche Ablauf einer Differenzierung gestaltet sich dabei wie folgt: In einem ersten Schritt werden die zu vergleichenden technischen Dokumente in ein internes Datenmodell eingelesen (vgl. [Weh04], S. 32). Ein Beispiel eines solchen internen Datenmodells ist in Abbildung 12 (in Anlehnung an [Kel06]) zu sehen. Innerhalb des Datenmodells ist ein Dokument als Graph repräsentiert, wobei die jeweiligen Elemente als getypte Knoten dargestellt werden. Weiterhin werden Abhängigkeiten und Beziehungen, die zwischen den Elementen existieren, durch getypte Kanten repräsentiert. Sonstige, über den Typ hinausgehende, Eigenschaften der Elemente werden als *Attribute* in Form einfacher *Key/Value*-Paare abgebildet. Das vorgestellte interne Datenmodell stellt nur eine Möglichkeit dar, wie sich Dokumente für eine Differenzanalyse modellieren lassen. Es ist allerdings wegen seines generischen Ansatzes als allgemeingültig anzusehen (vgl. [Tre07], S. 40). Für die vorliegende Arbeit wird die Begriffswelt auf Dokumentebene beibehalten und von Dokumenten, Elementtypen und Elementen gesprochen. Wie zu sehen ist, ist eine auf die Graphrepräsentation ausgerichtete Begriffswelt im vorliegenden Kontext als äquivalent anzusehen.

Der eigentliche Algorithmus zur Berechnung der Differenzen zwischen Dokumenten gliedert sich in zwei Phasen:

- Einer Hashingphase
- einer anschließenden Matchingphase

In der ersten Phase, der *Hashingphase*, werden die Hashwerte von allen Elementen aus den vorliegenden Dokumenten berechnet und verglichen. Weisen zwei Elemente den gleichen Hashwert auf, so erfolgt eine Zuordnung. Dabei gilt allerdings eine Einschränkung. Zuordnungen erfolgen einzig, wenn sie eindeutig sind. Sollten in einem Dokument zwei oder mehr Elemente identische Hashwerte aufweisen, so kann keine eindeutige Entscheidung über eine Korrespondenz getroffen werden. Alle Elemente eines Dokuments, für die nach Abschluss der Hashingphase noch kein Partner gefunden wurde, durchlaufen im Weiteren eine *Matchingphase*. Dabei werden die Elemente, getrennt nach Elementtypen, mittels Vergleichsfunktionen paarweise miteinander verglichen und jeweils der Ähnlichkeitswert einer Paarung berechnet. Übersteigt der Ähnlichkeitswert eine festgelegte Schwelle, so werden Elemente als grundsätzlich ähnlich zueinander angesehen. Sind die Ähnlichkeitswerte für alle möglichen Paarungen berechnet, erfolgt für die jeweils am Besten bewerteten Paarungen eine Zuordnung. Dabei gilt wieder die bereits für die Hashingphase getroffene Einschränkung, dass nur eindeutige Zuordnungen vorzunehmen sind.

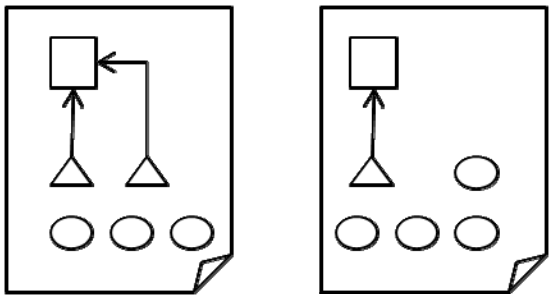


Abbildung 13: Symbolische Darstellung zweier Dokumente

Liegen zwischen bestimmten Elementtypen eines Dokuments *Teil-von-Beziehungen* vor, so ist dieser Sachverhalt insbesondere zu berücksichtigen. Die Problematik dabei ist in Abbildung 13 (in Anlehnung an [Tre07], S. 2) exemplarisch dargestellt. In den beiden abgebildeten Dokumenten kommen jeweils drei

verschiedene Elementtypen (*Quadrat*, *Dreieck* und *Kreis*) vor, wobei die Menge der Dreiecke einen Teil der Quadrate bilden. Damit eine Aussage über Ähnlichkeiten von in der Hierarchie höher stehenden Elementen (hier Quadrate) getroffen werden kann, sind dementsprechend zuerst die Ähnlichkeiten zwischen den Kindselementen (hier Dreiecke) zu untersuchen. Das Ergebnis dieser Ähnlichkeitsberechnung wird an die höher stehenden Elemente propagiert (*Bottom-Up-Phase*). Auf dieser Basis wird dann die Ähnlichkeit der übergeordneten Elemente untersucht und gegebenfalls Zuordnungen vorgenommen. Eine erfolgte Zuordnung zwischen zwei Elementen wird dabei innerhalb der hierarchischen Beziehung auch wieder an die Kindselemente propagiert (*Top-Down-Phase*) und im Folgenden deren Ähnlichkeit erneut (vgl. [Kel06], S. 3) berechnet. Dies ist nötig, da aufgrund gefundener Korrespondenzen Veränderungen der Ähnlichkeitswerte eintreten können. Die beiden beschriebenen Phasen werden solange im Wechsel ausgeführt, bis keine weiteren Zuordnungen mehr gefunden werden. Der schematische Ablauf der Matchingphase ist Abbildung 14 (in Anlehnung an [Weh04], S. 42) zu sehen.

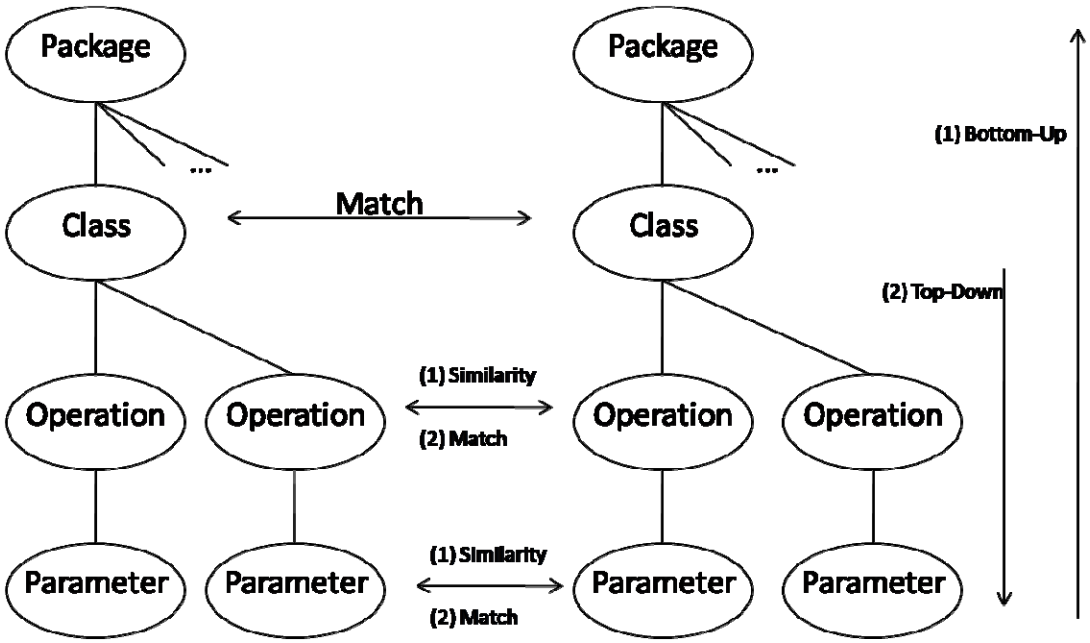


Abbildung 14: Schematischer Ablauf paarweiser Vergleich

Die zugrundeliegenden Kriterien, um eine Ähnlichkeit zwischen den Elementen zu berechnen, sind in Konfigurationsdateien definiert. Tabelle 1 zeigt exemplarisch

die Vergleichskriterien für Klassen aus der Domäne der UML-Klassendiagramme[18] (vgl. [Weh04], S. 60f).

Dabei wird für jedes Kriterium anhand der in der Spalte *ComparedItem* festgelegten Ähnlichkeitsfunktion ein Ähnlichkeitswert zwischen *0* (keine Ähnlichkeit) und *1* (Übereinstimmung) berechnet. Jede der berechneten Ähnlichkeiten wird dann mit dem für das jeweilige Vergleichskriterium in Spalte *W* definierten Gewicht multipliziert. Sollten für die Berechnung weitere Parameter nötig sein, so sind diese in der Spalte *Parameter* vermerkt. Die Summe aller Gewichte aus der Spalte *W* muss 1 ergeben.

Die Ähnlichkeit zweier Elemente ist durch die Summe der gewichteten Ähnlichkeitswerte der einzelnen Vergleichsfunktionen definiert. Überschreitet der berechnete Ähnlichkeitswert den in Spalte *t* festgelegten Schwellenwert, so ist von einer grundsätzlichen Ähnlichkeit der Elemente auszugehen.

Elementtyp	***t***	***ComparedItem***	***Parameter***	***W***
Class	0,4	CISimilarNames	-	0,4
		CIUnorderedCompartment	Operation	0,2
		CIUnorderedCompartment	Attribute	0,2
		CIUnorderedCompartment	Generalization	0,1
		CIMatchingParents	-	0,1

Tabelle 1: Vergleichskriterien für UML-Klassen

Nachdem die Phase der Berechnung der Differenzen abgeschlossen ist (vgl. [Weh04], S. 47f), erfolgt auf Basis der gefundenen Zuordnungen und Differenzen die Erstellung eines vereinheitlichtes Modells, welches in einem letzten Schritt wieder in ein XMI-Dokument geparst und gespeichert wird.

Die Laufzeitkomplexität des vorgestellten Algorithmus zur Berechnung von Differenzen wird von *Wehren* (vgl.[Weh04], S. 87ff) anhand der Laufzeit der *Matchingphase*, welche die aufwendigste Phase des Algorithmus darstellt, mit *O(|E1| x |E2| x Ccompare)* abgeschätzt. *E1* und *E2* repräsentieren dabei die Menge der Elemente in den zu vergleichenden Dokumenten, während *Ccompare* für die Kosten des Vergleichs zweier Elemente in der Matchingphase steht. Geht man von

[18] *Es werden von Wehren natürlich auch die Kriterien für die weitere Elementtypen aus der Domäne Klassendiagramme vorgestellt. Die entsprechenden Kriterien für Aktivitätsdiagramme sind in [Weh04] auf S. 65f. zu finden*

etwa gleich großen Dokumenten aus, so ergibt sich für den Algorithmus in etwa eine quadratische Laufzeitklasse.

Wie *Wehren* weiterhin feststellt (siehe [Weh04], S. 89) hat aber auch die Wahl der Vergleichsfunktionen einen großen Einfluss auf die reale Laufzeit des vorgestellten Algorithmus. Insbesondere bei großen Mengen von zu vergleichenden Elementen und einer Ähnlichkeitsberechnung anhand mehrerer, aufwendiger Kriterien erscheint deshalb der Einsatz eines Screenings lohnend. Die Idee hinter diesem Ansatz ist analog zu der bereits eingangs vorgestellten Intention (siehe *Kapitel 3.1 Motivation für ein Screening*) der Durchführung eines fingerprintprintbasierten Screenings zur Vermeidung komplexer Verfahren bei Suchanfragen auf molekularen Datenbanken.

4.2 Unterschiede der Einsatzgebiete

Wie bereits beschrieben, ist die Aufgabe eines Fingerprintverfahrens bei einer Anfrage auf einer molekularen Datenbank ein Screening nach aussichtreichen Kandidaten durchzuführen. Dazu wird zu jeder Anfrage diejenige Menge an chemischen Strukturen extrahiert, die auf Basis ihrer Fingerprints einen möglichen Treffer auf die Suchanfrage darstellt. Danach ist zu überprüfen, welche Elemente dieser Menge die gesuchte chemische Struktur enthalten. Die Fingerprints für chemische Strukturen werden dabei entweder mittels eines zuvor festgelegten Musterkatalogs oder anhand von Fragmenten aus dem jeweiligen Molekülgraph gebildet.

Überträgt man diesen in der Bioinformatik vorliegenden Anwendungsfall eins-zu-eins in den Kontext der Dokumentdifferenzierung, so wird der Frage nachgegangen, welche Dokumente aus einer gegebenen Menge ein gesuchtes Musterdokument enthalten. Die Bestimmung einer Kandidatenmenge würde in diesem Fall über Fingerprints mit dokumentspezifischen Merkmalen geschehen. Für einen Ansatz der Differenzbildung zwischen Dokumenten auf Basis ihrer Elemente, wie er in *Kapitel 4.1 Ein Algorithmus zur Berechnung von Dokumentdifferenzen* beschrieben ist, stellt ein derartiges Vorgehen allerdings keinen Gewinn dar.

Die im Kontext der Bioinformatik in einem Fingerprint abgebildeten Eigenschaften sind Charakteristika[19] der jeweiligen chemischen Struktur, respektive des Molekülgraphs. In dem vorgestellten Kontext der Differenzanalyse zwischen Dokumenten wird hingegen der Frage nachgegangen, ob eine Korrespondenz zwischen einem Element aus einem Dokument mit einem Element aus einem weiteren Dokument zu erkennen ist. Um ein effektives Screening von Elementen unter dem Gesichtspunkt dieser Fragestellung durchzuführen, reicht es natürlich nicht aus einen Fingerprint über charakteristische Merkmale der Dokumente zu erstellen. Vielmehr ist es nötig, dass jedes einzelne Element einen Fingerprint mit charakteristischen Merkmalen des jeweiligen Elementtyps aufweist. Dass im Rahmen einer Differenzbildung zwischen Dokumenten kein spezielles Musterelement *P* mehr auftritt, sondern vielmehr jedes Element bei einem paarweisen Vergleich als Muster fungiert, ändert selbstverständlich nichts an der grundsätzlichen Intention des Verfahrens.

Während Dokumente in der Regel eine Vielzahl verschiedener Ansatzpunkte für die Ermittlung von charakteristischen Eigenschaften bieten, ist dies für Elementtypen nicht zwingend gegeben. Deshalb ist weiterhin bei einem Einsatz des

[19] *Ein Charakteristikum sei an dieser Stelle als eine Eigenschaft, die sich für eine Differenzierung zweier Elemente nutzen lässt, definiert. Dies trifft nicht automatisch auf alle denkbaren Eigenschaften von Elementen zu.*

Verfahrens sicherzustellen, dass Elementtypen reichhaltig genug an charakteristischen Merkmalen sind. Nur dann kann ein effektives Selektieren von Kandidatenelementen erfolgen.

Ein weiteres Problem bei dem Transfer des Verfahrens liegt in der Abbildung von charakteristischen Eigenschaften eines Elements auf einen Fingerprint. In der Regel beruht die Berechnung eines Ähnlichkeitswerts in der Differenzanalyse von Dokumenten nicht auf fixen Merkmalen wie *besitzt einen Benzolring* (strukturelle Fingerprints) oder *besitzt ein Fragment* (Hashed Fingerprints), sondern auf Ähnlichkeiten von metrischen und lexikalischen Eigenschaften. Es ist deshalb insbesondere zu überlegen, wie diese beiden Arten von Eigenschaften in der bitorientierten Struktur eines Fingerprints abgebildet werden können. Wie später in dieser Arbeit gezeigt wird, ist es bei lexikalischen Merkmalen oftmals dafür nötig, von wichtigen Eigenschaften der zugrundeliegenden Zeichenkette zu abstrahieren. Auch weitere in der Differenzanalyse häufig genutzte Ähnlichkeitskriterien, wie beispielsweise Ähnlichkeiten zwischen benachbarten Elementen, lassen sich nicht oder nicht problemlos in einem Fingerprint abbilden. Ein fingerprintbasiertes Screening setzt vor der Berechnung der Ähnlichkeitswerte an und kann deshalb über die während der Differenzanalyse erzielten Ergebnisse keine Annahmen treffen.

Aus den in diesem Kapitel erörterten Gründen folgt auch, dass ein Ansatz auf Basis eines an Hashed Fingerprints orientierten Verfahrens als nicht geeignet für den gegebenen Kontext anzusehen ist. Eine Abbildung von Fragmenten in einem Fingerprint eignet sich nur bedingt für die charakteristische Beschreibung eines einzelnen Elements. Ein Fragment besteht per Definition aus Verbindungen und enthält somit implizit Informationen, die die Umgebung eines Elements näher beschreiben. Kommt es zu einer Änderung der Umgebung, so hat dies folglich direkte Auswirkungen auf den Fingerprint. Werden dabei, wie bei Hashed Fingerpints, auch Verbindungen verschiedener Länge betrachtet, so schlägt sich eine Änderung mitunter wiederholt in einem Fingerprint nieder. Diese Problematik ist in Abbildung 15 anhand zweier Dokumente dargestellt, die wieder aus den Elementtypen Quadrat, Kreis und Dreieck bestehen. Die beiden Dokumente unterscheiden sich zwar nur in einem Element und sind ansonsten als identisch anzusehen, aber wie in dem unteren Teil der Abbildung dargestellt ist, haben sich infolgedessen fast alle in einem Hashed Fingerprint des Elements *Kreis* abgebildeten Fragmente geändert. Untersucht man die Auswirkung der dargestellten Änderung auf das Element *Quadrat,* so ist sogar festzustellen, dass sich alle betrachteten Fragmente verändert haben. Wenn man weiter bedenkt, dass ein Element aus der Nachbarschaft nicht einmal zwingend ein Charakteristikum des betrachteten Elements darstellt, ist sofort verständlich, dass dieser Ansatz für das gegebene Anwendungsgebiet nur bedingt geeignet ist.

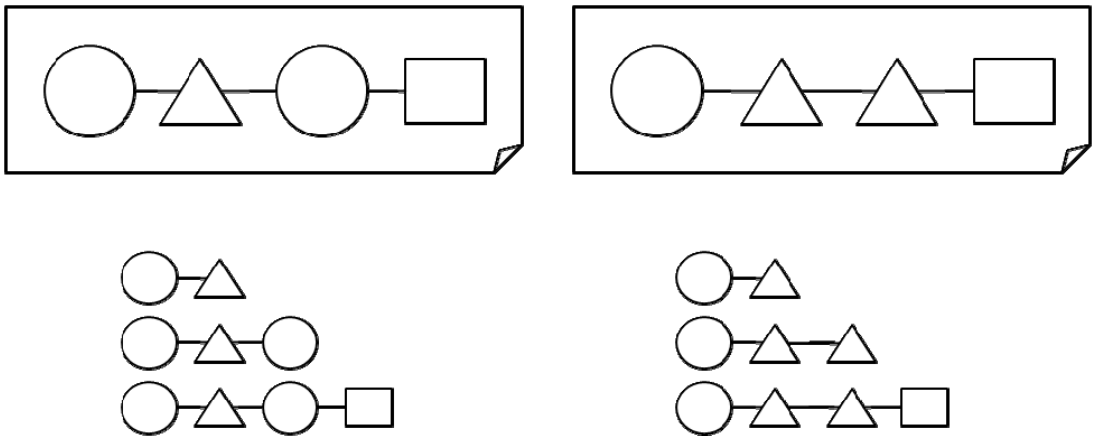


Abbildung 15: Veränderung der Nachbarschaft

In speziellen Fällen kann es allerdings sehr wohl sinnvoll sein, Elemente aus der Nachbarschaft und deren Eigenschaften als Charakteristika für eine genauere Beschreibung eines Elements heranzuziehen. Ein Beispiel dafür aus der Domäne der UML-Klassendiagramme sind Klassen, wo zugehörige Methoden als Merkmale des Klassenelements aufgefasst werden können (vgl. [Lan99], S. 11). Da dies allerdings die Ausnahme darstellt, ist eine Abbildung dieser Sachverhalte über Merkmale, wie sie von strukturellen Fingerprints genutzt werden, sinnvoller als den Spezialfall mittels eines Hashed Fingerprints zu verallgemeinern.

Der wichtigste Unterschied zwischen den beiden Anwendungsgebieten liegt allerdings in der Genauigkeit, mit der die Kandidatenmenge gebildet werden kann. Bei einer Suche in einer molekularen Datenbank kann davon ausgegangen werden, dass aussichtreiche Kandidaten in ihren Fingerprints mindestens dieselben Merkmale aufweisen, wie der Fingerprint über die Suchstruktur. Auf Grund dessen lässt sich bei einer solchen Suche nicht nur mittels einfacher Überprüfung der *Teil-von*-Beziehung eine dementsprechend hohe Prozentzahl (Idealerweise mehr als 99%. (vgl. [Lea03], S. 9)) der Elemente durch ein Screening ausschließen, sondern es ist auch sichergestellt, dass einzig Elemente gefiltert werden, die definitiv nicht auf die Suchanfrage passen. In der Differenzanalyse von Dokumenten ist allerdings implizit davon auszugehen, dass potentielle Kandidaten durchaus Unterschiede in ihren Eigenschaften und somit auch ihren Fingerprints aufweisen. Dies hat zur Folge, dass bei der Bildung der Kandidatenmenge auf die bereits vorgestellten Ähnlichkeitsmaße für Fingerprints zurückgegriffen werden muss. Die wesentliche gravierendere Implikation ist allerdings, dass aus diesem Grund nicht mehr sichergestellt werden kann, dass einzig Kandidaten gefiltert werden, die definitiv keine Aussicht auf einen hohen Ähnlichkeitswert besitzen. Aufgrund der hieraus resultierenden Unschärfe ist das vorgestellte Verfahren dementsprechend als

heuristischer Ansatz zu verstehen. Verstärkt wird dieses Problem dadurch, dass sich nicht alle Ähnlichkeitskriterien problemlos in einem Fingerprint darstellen lassen. Durch Bildung einer Kandidatenmenge in der Differenzanalyse kann demnach kein optimales Ergebnis mehr garantieren werden.

Abschließend ist festzuhalten, dass die Nutzung eines fingerprintbasierten Screenings in der Differenzanalyse von Dokumenten im Vergleich zu dem originären Einsatzgebiet sich insgesamt wesentlich schwieriger gestaltet. Durch den elementorientierten Ansatz des vorgestellten Algorithmus zur Differenzanalyse ergibt sich, dass eine Erstellung eines Fingerprint für das gesamte Dokument nicht mehr ausreicht. Stattdessen ist ein Fingerprint für jedes Element innerhalb des Dokuments zu berechnen. Weiterhin wurde gezeigt, dass ein Ansatz angelehnt an Hashed Fingerprints sich als nicht sinnvoll erweist. Abgebildete Fragmente können sich zum einen leicht ändern und sind zum anderen nicht zwingend auch charakteristische Merkmale einzelner Elemente. Es konnte weiterhin festgestellt werden, dass die Abbildung von Eigenschaften und Ähnlichkeitskriterien in die bitorientiere Struktur eines Fingerprints problematisch ist, wodurch die sinnvolle Einschränkung der Kandidatenmengen erschwert wird. Das größte Manko des Ansatzes ist allerdings, dass durch Bildung von Kandidatenmengen keine optimalen Ergebnisse mehr garantiert werden können.

Die grundsätzliche Idee, ein Screening einer Kandidatenmenge durchzuführen, ist allerdings sehr wohl auch in der Dokumentdifferenzierung von Interesse. Insbesondere bei einer großen Anzahl von Elementen eines Typs ist auf Grund der verfahrensimpliziten quadratischen Laufzeitkomplexität mitunter eine immense Anzahl an Ähnlichkeitsberechnungen durchzuführen. Kann für ein Element eines derart dominierenden Elementtyps effizient entschieden werden, für welche Kandidatenelemente eine Berechnung der genauen Ähnlichkeitswerte vorzunehmen ist, so ist durch die Einsparung an durchzuführenden Vergleichsfunktionen eine Verbesserung der realen Laufzeit zu erwarten.

Es ist an dieser Stelle abzusehen, dass der Einsatz eines fingerprintbasierten Screenings insbesondere für Modelltypen interessant erscheint, die bestimmte Eigenschaften aufweisen. Je häufiger in Dokumenten eines zugrundeliegenden Modells Elemente eines bestimmten Typs auftreten, desto interessanter ist auf Grund der quadratischen Laufzeit des Algorithmus die Filterung einer geeigneten Kandidatenmenge. Dabei ist allerdings zu beachten, dass die Elemente eines solchen dominierenden Typs eine genügend große Anzahl an charakteristischen Merkmalen aufweisen müssen, anhand derer ein effektives Screening erfolgen kann. Elemente mit identischen Eigenschaften, sowie Elemente bei denen nicht ersichtlich ist,

welche Eigenschaften als charakteristisch anzusehen sind, sind für das vorgestellte Verfahren problematisch. Erstgenannte weisen keine Unterschiede in ihren Fingerprints auf, während bei Letztgenannten nicht klar ist, welche Eigenschaften abzubilden sind. In beiden Fällen kann der Einsatz eines Screenings auf Basis von Fingerprints zu keinem Erfolg führen. Weiterhin ist anzunehmen, dass mit steigender Komplexität der zur Bestimmung der Ähnlichkeit genutzten Kriterien auch die Aussicht auf eine Einsparung an Laufzeit steigt.

4.3 Implementierung des Fingerprintverfahrens

In diesem Kapitel wird eine Implementierung eines fingerprintbasierten Screenings skizziert. Grundlage hierfür ist die Implementierung des in *Kapitel 4.1 Ein Algorithmus zur Berechnung von Dokumentdifferenzen* Algorithmus, welche im Folgenden als *SiDiff* referenziert wird. Zielsetzung der vorgeschlagenen Implementierung eines Fingerprintverfahrens ist die schnelle Nutzung des Verfahrens bei gleichzeitig möglichst hoher Flexibilität.

Abbildung eines Fingerprints

Als Kernstück der Implementierung ist die Klasse *BasicFingerprint* (vgl. Abbildung 16) anzusehen, welche aus der Java-Klasse[20] *BitSet* abgeleitet ist. Neben den geerbten Methoden sind zusätzliche Funktionalitäten, beispielsweise für eine Ausgabe des Fingerprints oder um bei Bedarf eine Faltung vorzunehmen, implementiert. Um sicherzustellen, dass jedes Element, das innerhalb des internen Datenmodells als Knoten abgebildete ist, einen Fingerprint besitzt, wurde die SiDiff-Klasse *Node* um ein Attribut *Fingerprint* der Klasse *BasicFingerprint* erweitert.

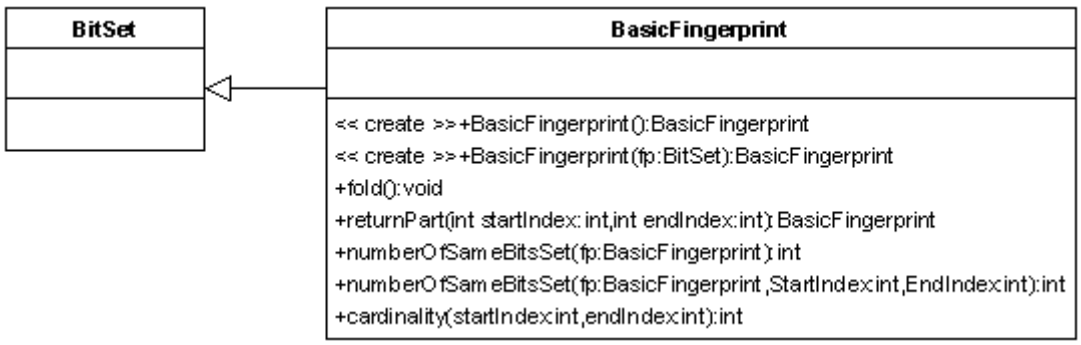


Abbildung 16: Klasse BasicFingerprint

Somit ist für jedes Element die Möglichkeit gegeben, einen Fingerprint über dessen charakteristischen Merkmale zu hinterlegen. Im Weiteren ist es natürlich

[20] *Da der von Wehren vorgestellte Algorithmus in der Programmiersprache Java implementiert ist, war die Entscheidung, welche Programmiersprache zur Implementierung des Fingerprintverfahren zu nutzen ist, obsolet.*

noch nötig festzulegen, wie zu prüfende Merkmale beschrieben und einzelne Elemente auf die beschriebenen Merkmale hin untersucht werden können.

Überprüfung eines Merkmals

Die Basis für die Überprüfung, ob ein Element ein bestimmtes Merkmal erfüllt, stellt die abstrakte Klasse *IsMatch* dar. Jede Implementierung dieser Klasse erbt Methoden zum Setzen der Bitstellen innerhalb eines Fingerprints, sowie einen Parameter *mapping*. Dieser Parameter gibt die Bitstelle[21] an, die das geprüfte Merkmal im Weiteren repräsentiert. Die eigentliche Funktionalität der Prüfung geschieht in der Implementierung der abstrakten Methode *isMatch(Node n)*. Dabei wird das zu überprüfende Element übergeben, auf das innerhalb der Methode spezifizierte Merkmal getestet und gegebenenfalls die entsprechende Bitposition gesetzt. Sollten für die Überprüfung eines Merkmals weitere Parameter nötig sein, so kann eine Implementierung der abstrakten *IsMatch*-Klasse selbstverständlich im Sinne des objektorientierten Konzeptes der Vererbung erweitert werden.

Der Quellcode der Klasse NumberOfAttributes (siehe Abbildung 17) zeigt beispielhaft eine Implementierung der *IsMatch*-Klasse. *NumberOfAttributes* zählt dabei die Anzahl der als Key/Value-Paare abgebildeten zusätzlichen Attribute, die ein übergebenes Element *n* aufweist. Neben *mapping* ist für die Abbildung dieses Merkmals noch der Parameter *range* definiert. Dieser legt die maximale Anzahl an Attributen fest, die noch in einem Fingerprint abzubilden ist. Ist die Anzahl der Attribute eines Elements größer als *range*, kommt es zu keiner Abbildung. Ein derartiger Mechanismus dient dazu, das Auftreten beliebig langer Fingerprints zu verhindern und das Abbilden von Merkmalen auf bestimmte Bereiche des Fingerprints zu beschränken.

Nachdem nun mit der vorgestellten IsMatch-Klasse auch ein Konzept eingeführt ist, mit dem sich Merkmale überprüfen lassen, bleibt noch zu definieren, wie die Erstellung eines Musterkatlogs erfolgen kann. Dazu wird ein Ansatz vorgeschlagen, die zu prüfenden Muster in einer XML-Konfigurationsdatei zu definieren.

[21] *Wenn eine Eigenschaft getestet wird, die nicht in einer einzelnen Bitstelle repräsentiert werden kann, so gibt „mapping“ die Basisposition an, ab der das Merkmal in dem Fingerprint abgebildet ist.*

```
1   public class NumberOfAttributes extends IsMatch {
2
3         private int range;
4
5         public NumberOfAttributes(int mapping, int range) {
6         setMapping(mapping);
7         this.range = range;
8         }
9
10        public void isMatch(Node n) {
11              int numberOfAttributes = n.getAttributes().size();
12
13              if (numberOfAttributes < range)
14                        setBit(n, mapping + numberOfAttributes);
15              }
16
17  }
```

Abbildung 17: Quellcode NumberOfAttributes

Bildung eines Musterkatalogs

Die in einem Fingerprint abzubildenden Merkmale mittels einer XML-Konfigurationsdatei zu beschreiben, ist eine einfache und flexible Lösung für die Definition eines Musterkatalogs. Eine derartige Konfigurationsdatei legt die in einem Fingerprint repräsentierten Merkmale ähnlich einer Blaupause fest und wird im Weiteren auch als *Blueprint* bezeichnet.

Die einzelnen XML-Elemente der Konfigurationsdatei spezifizieren dabei, welche Merkmale zu überprüfen sind. Die für eine Überprüfung nötigen Parameter werden mit Hilfe von *Attributes* innerhalb des jeweiligen Elements angegeben. Ein derartiges Vorgehen ermöglicht bei geeigneter Implementierung der Merkmale eine flexible Nutzung der Abfrageobjekte mittels Parametrisierung.

Abbildung 18 zeigt dies beispielhaft an dem Merkmal *NumberOfNeighborNodes*, welches die Anzahl der benachbarten Knoten zählt und in dem Fingerprint des Elements abbildet. Dabei wird über das Attribut *nodetype* festgelegt, für welchen Elementtyp das Merkmal abzubilden ist. In diesem Fall wird das Merkmal für Klassen abgefragt. Weiterhin ist durch den Parameter *mapping* festgelegt, dass die Abbildung der Anzahl an der 110 Bitposition, welche einem Wert von 0 entspricht, beginnt. Der Parameter *range* gibt den maximalen, noch abzubildenden Wert wieder. Der Parameter *direction* legt schließlich fest, Kanten welcher Richtung zu den Nachbarknoten betrachtet werden sollen. In diesem Fall signalisiert der Parameter *any*, dass alle Knoten der Nachbarschaft unabhängig von der Richtung der Kante zu berücksichtigen sind. Eine *Document Type Definition,*

gegen die ein Musterkatalog auf Korrektheit geprüft werden kann, ist in *Anhang A I: DTD* abgebildet.

```
<NumberOfNeighborNodes
      nodetype="class"
      mapping="110"
      range="50"
      direction="any"/>
```

Abbildung 18: Beispiel Merkmal NumberOfNeighborNodes

Ablauf der Erstellung eines Fingerprints

Die jeweilige XML-Konfigurationsdatei wird intern mit Hilfe eines *SAX-Parsers* ausgelesen. Jedes der Wurzel untergelagerte Element repräsentiert dabei ein in dem Fingerprint abzubildendes Merkmal. Für jedes der beschriebenen Merkmale existiert eine entsprechende Implementierung der Klasse *IsMatch* gleichen Namens. Der SAX-Parser liest der Reihe nach alle in der XML-Konfigurationsdatei definierten Merkmale inklusiv Parametern ein, erstellt die entsprechenden Objekte zur Abfrage und speichert sie - hinterlegt nach dem zu testenden Elementtyp - in einer geeigneten Datenstruktur ab. Ist die Überprüfung eines Merkmals nicht auf einen speziellen Elementtyp, sondern für alle Elemente eines Dokuments unabhängig ihres Typs durchzuführen, so kann dies über den Pseudotyp *any* abgebildet werden. Auftretende Fehler werden, um den weiteren Ablauf des Verfahrens sicherzustellen, abgefangen und die entsprechende Fehlermeldung in einer Log-Datei vermerkt. Nachdem das Einlesen der Konfigurationsdatei und die Erzeugung der Abfrageobjekte beendet sind, wird im nächsten Schritt jeweils über die Elemente der zu differenzierenden Dokumente iteriert. Für jedes Element wird der Elementtyp erfragt und - sofern zu überprüfende Merkmale für einen Typen vorliegen - die entsprechenden *IsMatch*-Objekte aus der Datenstruktur extrahiert. Liegen Merkmale vor, wird das jeweilige Element auf deren Erfüllung geprüft und entsprechend dem Ergebnis der Prüfung der Fingerprint des Elements erstellt.

Die Stärke des vorgestellten Ansatzes auf Basis von XML-Konfigurationsdateien liegt in der Tatsache, dass diese schnell und einfach angepasst werden können. So ist es durch diesen Ansatz möglich, effizient Tests mit verschiedenen Abfrageobjekten oder Parameterwerten durchzuführen, ohne das hierfür eine Zeile Quelltext geändert werden braucht und somit das System auch nicht erneut kompiliert werden muss.

Screening der Elemente

Nachdem die Alternativen für die Erstellung eines Fingerprints vorgestellt wurden, ist für einen Einsatz des Verfahrens letztendlich nur noch ein Ansatz zu definieren, der einen fingerprintbasierten Ähnlichkeitswert zwischen den Elementen der zu vergleichenden Dokumente berechnet. Ein so errechneter Wert kann im Rahmen einer Schwellenwertanalyse (vgl. *Kapitel 3.4.2 Ähnlichkeitsabfragen und Nachbarschaft*) zur Bildung einer Kandidatenmenge genutzt werden, um Kandidaten mit einer voraussichtlich niedrigen Ähnlichkeit zu filtern.

Ein in SiDiff bereits implementierter Mechanismus bietet für die beschriebene Funktionalität eine einfache Lösung. So können bei einer Dokumentdifferenzierung für den Vergleich von Elementen weitere Nebenbedingungen, sogenannte *Constraints*, festgelegt werden. Mit Hilfe einer Constraint lässt sich eine Bedingung definieren, welche von zwei zu vergleichenden Elementen erfüllt sein muss, damit diese im Weiteren einer genauen Berechnung ihrer Ähnlichkeit unterzogen werden. Um eine solche Constraint zu definieren ist einfach die abstrakte Klasse *Constraint*, insbesondere deren Methode *public boolean checkConstraint(Node a, Node b),* zu implementieren. Die übergebenen Elemente *a* und *b* werden auf die durch die jeweilige Implementierung der Methode festgelegten Bedingung geprüft und, sollten sie die Bedingung nicht erfüllen, von der weiteren Ähnlichkeitsberechnung ausgeschlossen. In diesem Fall wird von vollständig unterschiedlichen Elementen mit einem Ähnlichkeitswert von *0* ausgegangen. Es ist sofort ersichtlich, dass sich dieser Mechanismus bestens für die eingangs geforderte Funktionalität einsetzen lässt.

```
1   public class Tanimoto extends Constraint {
2
3         private double threshold;
4
5         public Tanimoto(double threshold) {
6               super(null,null);
7               this.threshold = threshold;
8         }
9
10        public boolean checkConstraint(Node nodeA, Node nodeB) {
11
12              if (nodeA.getFP().cardinality() == 0
13                    && nodeB.getFP().cardinality() == 0) return true;
14
15
16              BasicFingerprint helpFP =
17                    (BasicFingerprint) nodeA.getFP().clone();
18              helpFP.and(nodeB.getFP());
19
20
21              double c = (double) helpFP.cardinality();
22              double a = (double) nodeA.getFP().cardinality();
23              double b = (double) nodeB.getFP().cardinality();
24
25              if ((c / (a + b -c)) >= threshold)
26                          return true;
27
28        return false;
29        }
30  }
```

Abbildung 19: Quellcode Tanimotokoeffizient

Alles was dafür nötig ist, ist die Implementierung einer passenden Constraint–Klasse vorzunehmen. Innerhalb der Implementierung der *checkConstraint()*-Methode kann dann der Ähnlichkeitswert über die Fingerprints der übergebenen Elemente mittels des gewünschten Ähnlichkeitsmaßes errechnet und gegen einen festgelegten Grenzwert verglichen werden.

Ein Beispiel einer solchen Implementierung zeigt der Java-Quelltext in Abbildung 19, in dem der bereits in *Kapitel 3.4.1 Ähnlichkeits- und Distanzmaße* vorgestellte Tanimotokoeffizient zum Einsatz kommt. Sollte bei einer ersten Überprüfung festgestellt werden, dass keine charakteristischen Eigenschaften der Elemente in ihren Fingerprints repräsentiert sind, so ist automatisch von einer grundsätzlichen Ähnlichkeit auszugehen. Sollte dieser Fall nicht vorliegen wird folgend der Tanimotokoeffizient zwischen den Fingerprints ermittelt und in einem letzten Schritt mit dem Grenzwert *threshold* verglichen. Übersteigt der errechnete

Wert den Grenzwert, liegt eine genügend hohe Ähnlichkeit der Elemente vor und es ist deren genaue Ähnlichkeit zu untersuchen.

Laufzeitabschätzung der vorgeschlagenen Implementierung

Abschließend noch eine kurze Laufzeitabschätzung des vorgeschlagenen Verfahrens. Das Einlesen der abzubildenden Merkmale aus der XML-Konfigurationsdatei erfolgt sequentiell und die Laufzeit ist somit linear abhängig von der Anzahl der in der Datei definierten Muster *M*. Für die Phase der Erstellung der Fingerprints ist eine Laufzeitkomplexität von *O((|E1| + |E2|) x CFPgenerate)* anzusetzen. *E1* und *E2* repräsentieren dabei die Menge an Elementen aus dem jeweiligen Dokument, während *CFPgenerate* für die durchschnittliche Komplexität der Erstellung eines Fingerprints steht. Wie beschrieben ist jedoch plausibel davon auszugehen, dass unter der gegebenen Intention des Verfahrens keine Berechnung von Merkmalen mittels aufwendiger Algorithmen erfolgt.

Wie zu sehen ist, verändert der Einsatz eines fingerprintbasierten Screening also nicht die mit *O(|E1| x |E2| x Ccompare)* (siehe [Weh04], S. 89) angegebene Laufzeitklasse des von *Wehren* vorgestellten Algorithmus. Sehr wohl kann aber ein Screening den Aufwand für den Vergleich zweier Elemente mit wenig Aussicht auf Korrespondenz einsparen, sofern sinnvolle Charakteristika für eine derartige Selektion feststellbar sind.

4.4 Abbildung von Merkmalen

Der folgende Abschnitt beschäftigt sich eingehend mit den verschiedenen Möglichkeiten zur Abbildung von Eigenschaften innerhalb eines Fingerprints. Dabei wird insbesondere auf verschiedene Abbildungsmöglichkeiten und deren Auswirkungen auf die Berechnung eines fingerprintbasierten Ähnlichkeitswertes eingegangen. Es lässt sich dabei allgemein zwischen drei verschiedenen Ausprägungen von charakteristischen Eigenschaften, die im Kontext der Differenzbildung zwischen Dokumenten grundsätzlich für die Durchführung eines Screening interessant sind, unterscheiden:

- Einfache Eigenschaften
- Metrische Eigenschaften
- Lexikalische Eigenschaften

Bevor im Weiteren auf jede der Ausprägungen genauer eingegangen wird, werden einige allgemeine Aspekte der Problematik einer Abbildung von Eigenschaften in einem Fingerprint näher erläutert.

Betrachtet man ein fingerprintbasiertes Screening auf Basis des vorgeschlagenen Verfahrens, so ergeben sich zwei wichtige Eigenschaften für eine Abbildung von Merkmalen:

- Identische Bitpositionen repräsentieren identische Merkmale
- Jede Bitposition ist bei der Berechnung der Ähnlichkeit gleichgewichtet

Die beiden Eigenschaften resultieren aus der Tatsache, dass die vorgestellten Ähnlichkeitsmaße einen Ähnlichkeitswert auf Basis der gesetzten und nicht gesetzten Bitstellen berechnen, insbesondere wenn deren Ausprägung in beiden Fingerprints identisch ist. Daraus folgt, dass eine Abbildung von Merkmalen, besonders wenn es sich dabei um lexikalische oder metrische Eigenschaften handelt, auf Basis ihrer binären Darstellung nicht sinnvoll ist.

Betrachtet man beispielsweise die binäre Codierung der Zahlen *1* [*=01*] und *3* [*=11*], so ist festzustellen, dass diese wegen der auftretenden identischen Bitposition eine Ähnlichkeit aufweisen. Allerdings besitzt die Zahl *1* auf Basis ihrer Binärcodierung keine Ähnlichkeit zu der Zahl *2* [*=10*] aufweist. Weiterhin fällt eine binäre Zahl, bei deren Codierung viele Bitstellen auf *1* gesetzt sind, bei der

Berechnung eines Ähnlichkeitswertes wesentlich stärker ins Gewicht. Führt man sich vor Augen, dass die binäre Codierung der Zahl 63 [=*0111111*] im Vergleich zu der binären Codierung des Zahlenwert 64 [=*1000000*] eine wesentlich höhere Anzahl an repräsentierten Merkmalen aufweist wird ersichtlich, dass Ansätze einer Abbildung von Merkmalen auf Basis ihrer Binärcodierung zu keinen brauchbaren Ergebnissen führen werden. Weiterhin kommt es bei einer binären Abbildung von lexikalischen Eigenschaften unter anderem zu der bereits in *Kapitel 2.1 Sequence Alignment* beschriebene Problematik einer impliziten Annahme über eine bereits erfolgte Ausrichtung der Zeichenketten.

4.4.1 Einfache Eigenschaften

Die Abbildung definierter Eigenschaften als Einzelmerkmale ist die am einfachsten innerhalb eines Fingerprints abzubildende Art von Merkmalen. Analog zu den in *Kapitel 3.2 Strukturelle Fingerprints* vorgestellten Merkmalen repräsentiert eine Bitstelle ein definiertes Merkmal und deren Ausprägung gibt Auskunft über dessen Präsenz. In der Regel wird dabei für eine sinnvolle Auswahl der abzubildenden Merkmale eine genauere Kenntnis der vorliegenden Domäne benötigt. Die Sichtbarkeit eines Elements in einem UML-Diagramm kann als Beispiel eines derartigen Merkmals angesehen werden. Dabei wird jeder der vier möglichen Ausprägungen der Sichtbarkeit (*public, private, protected oder package)* jeweils eine spezifische Bitposition innerhalb des Fingerprints zugeordnet und entsprechend der vorliegenden Ausprägung eines betrachteten Elements gesetzt.

4.4.2 Metrische Eigenschaften

Unter dem Begriff *Metrische Eigenschaften* sind Eigenschaften von Elementen zu verstehen, die sich als Zahlenwerte ausdrücken lassen. Besonders häufig kommt diese Art Eigenschaft für die Domäne der Klassendiagramme zum Einsatz, für die der Literatur eine Vielzahl an verschiedenen Softwaremetriken (siehe *5.2.3 Klassendiagramme*) vorgeschlagen ist.

Dass sich die Abbildung einer metrischen Eigenschaft in Form ihrer Binärcodierung als nicht geeigneter Ansatz erweist, wurde eingangs bereits gezeigt. Deshalb werden folgend zwei verschiedene Ansätze zur Abbildung besprochen.

Die zwei Ansätze sind dabei im Einzelnen:

- Direkte Codierung eines Wertebereiches
- Gestaffelte Abbildung

Direkte Codierung eines Wertebereiches

Eine einfache Möglichkeit für die Abbildung metrischer Eigenschaften ist es, eine Bitposition innerhalb des Fingerprints als Startindex festzulegen, der einen bestimmten Wert x[22] repräsentiert. Ausgehend von diesem Index entspricht die nächste Bitposition dabei dem um 1 inkrementieren Startwert und die *i-te* Bitposition, ausgehend von Startindex, repräsentiert demnach den Wert $x+i$.. Eine derartige Codierung des Wertes (siehe auch Abbildung 20 *Direkte Codierun*g) hat sich für die innerhalb der betrachteten Anwendungsgebiete auftretenden metrischen Indizes als ausreichend herausgestellt. Wird eine derartige Abbildung eingesetzt, so sind allerdings zwei Dinge zu bedenken:

Zum einen eignet sich diese Art der Abbildung nur bedingt für metrische Eigenschaften die sehr große Werte annehmen können. Je größer dabei die abzubildenden Werte werden können, desto mehr Bitpositionen sind in einem Fingerprint vorzuhalten. Es gibt allerdings verschiedene Möglichkeiten dem entgegenzuwirken. Beispielsweise kann durch eine, gegebenenfalls mehrfache, Faltung der Fingerprints auf eine vorgegebene Länge gebracht werden. Ist für eine metrische Eigenschaft nur in Ausnahmen ein Auftreten größerer Werte zu erwarten, kann mittels eines Parameters *max* der größte in einem Fingerprint abzubildende Wert definiert werden. Es ist zusätzlich denkbar, dass metrische Eigenschaften, deren Werte das festgelegte Maximum überschreiten, als einfache Eigenschaft der Art *besitzt mehr als* aufgefasst und entsprechend Abgebildet werden.

Weiterhin ist zu bedenken, dass unter Einsatz der vorgestellten Ähnlichkeitsmaße bei einer derartigen Abbildung jede Änderung des abgebildeten Wertes – unabhängig davon wie stark sich der Wert ändert - als Unterschied angesehen und dementsprechend in die Berechnung des Ähnlichkeitswertes einfließt.

[22] *In den betrachteten Anwendungsgebieten entspricht dies in der Regel dem Wert 0. Es ist aber natürlich möglich auch andere Zahlenbereiche auf diese Art in einem Fingerprint zu codieren.*

Als eine Erweiterung der vorgestellten direkten Codierung ist es natürlich auch denkbar, dass eine Bitposition statt eines festen Zahlenwertes einen Ausschnitt des abzubildenden Wertebereichs repräsentiert. Auf diese Weise lassen sich größere Wertebereiche platzsparend in einem Fingerprint abbilden und die für die direkte Codierung angesprochene Problematik einer Änderung des abgebildeten Wertes tritt nur auf, wenn diese über die Grenzen des Wertebereichs einer Bitposition hinaus erfolgt. Ein Beispiel hierfür ist in Abbildung 20 *Direkte Codierung 2* zu sehen, wobei jede Bitstelle einen Bereich von 5 Werten abdeckt. Geht man davon aus, dass das erste Bit eine 0 repräsentiert, so ist für eine Abbildung des Werts 10 die dritte Bitposition zu setzen.

Gestaffelte Abbildung einer metrischen Eigenschaft

Es findet sich in der Literatur für strukturelle Fingerprints (siehe [Day08]) ein weiterer Vorschlag für die Abbildung von metrischen Eigenschaften. Dabei wird der Wert einer metrischen Eigenschaft (für das folgende Beispiel als *mE* bezeichnet) nicht direkt codiert, sondern Bitpositionen innerhalb des Fingerprints geben Auskunft darüber, ob der Wert größer oder kleiner einer bestimmten Grenze ist. So ist es beispielsweise denkbar, dass eine Bitposition die Eigenschaft *mE>0* repräsentiert und die folgende Bitposition die Eigenschaft *mE>2*. Auf diese Weise wird eine gestaffelte Abbildung der Merkmale vorgenommen. Dabei ist allerdings zu bedenken, dass eine solche Staffelung bei einer Berechnung eines Ähnlichkeitswertes stärker ins Gewicht fällt, da eine Eigenschaft auf mehrere Bitpositionen abgebildet und somit auch mehrfach präsent ist. Diese Art der Codierung ist in Abbildung 20 *Gestaffelte Abbildung* zu sehen, wobei die erste Bitstelle die Bedeutung *mindestens zwei* besitzt und bei jeder weiteren Bitstelle der Vergleichswert um den Wert 2 inkrementiert wird.

```
Abbildung des Zahlenwerts 10

Binäre Codierung            00000001010
Direkte Codierung           00000000001
Direkte Codierung 2         00100000000
Gestaffelte Abbildung       11111000000
```

Abbildung 20: Abbildung metrische Eigenschaften

Wie gezeigt ergeben sich bei der Abbildung metrischer Eigenschaften innerhalb eines Fingerprintverfahrens zwei grundlegende Problematiken. Der Ansatz einer direkten Codierung führt zu der Problematik, dass jede Veränderung des abzubildenden Wertes zu einer Änderung der gesetzten Bitstelle führt und somit keine Ähnlichkeit mehr erkannt wird. Der Einsatz einer indirekten Codierung eines

Wertebereichs kann diesem Effekt zwar entgegenwirken, ihn aber nicht völlig auflösen. Die zweite Problematik tritt bei Verfahren auf, die einen Zahlenwert auf mehr als eine Bitstelle codieren und somit mitunter die Berechnung eines Ähnlichkeitswertes verfälschen.

4.4.3 Lexikalische Eigenschaften

Neben metrischen Eigenschaften von Elementen stützt sich die Differenzbildung zwischen Dokumenten insbesondere (siehe [Tre07], S. 43) auf die Ähnlichkeit lexikalischer Eigenschaften, beispielsweise Namen. Aus diesem Grund ist im Rahmen eines Screening eine geeignete Abbildung einer Zeichenkette von großem Interesse. Es wurde eingangs schon gezeigt, dass eine direkte Abbildung auf Basis einer binären Repräsentation der lexikalischen Eigenschaft nicht sinnvoll ist. Im Folgenden werden drei weitere Möglichkeiten zur Abbildung einer lexikalischen Eigenschaft vorgestellt und auf deren Vor- bzw. Nachteile genauer eingegangen.

Die drei Möglichkeiten der Abbildung lexikalische Eigenschaften im Einzelnen sind:

- Die Länge der Zeichenkette
- Abbildung der Buchstaben der Zeichenkette
- Abbildung der Zeichenkette als definiertes Merkmal

Länge einer Zeichenkette

Eine erste einfache Möglichkeit einer abstrakten Repräsentation von Zeichenketten ist es, die Anzahl ihrer Zeichen als metrisches Merkmal aufzufassen und mittels eines der vorgestellten Ansätze innerhalb des Fingerprints zu codieren. Es ist allerdings sofort ersichtlich, dass dieser Ansatz nicht ausreicht um eine lexikalische Eigenschaft sinnvoll abzubilden. Es wird hierbei komplett von den jeweils auftretenden Zeichen sowie deren Reihenfolge abstrahiert. Dies ist ein gravierendes Manko und macht diese Art der Darstellung für eine charakteristische Beschreibung einer einzelnen lexikalischen Eigenschaft ungeeignet.

Sollte ein Element allerdings mehrere lexikalische Eigenschaften aufweisen, die sich als charakteristische Merkmale für eine Differenzierung interpretieren lassen, so kann deren Länge sehr wohl sinnvoll für den Einsatz eines fingerprintbasierten Screenings genutzt werden. Ein Beispiel eines solches Merkmals aus der Domäne der UML-Klassendiagramme sind die Namen der Methoden einer Klasse. Da plausibel anzunehmen ist, dass es zwischen einzelnen Ausprägungen einer Klasse zu

Unterschieden sowohl in der Anzahl der Methoden als auch in der lexikalische Eigenschaft der jeweiligen Methodennamen kommt, ist es nicht möglich die Methodennamen direkt in dem Fingerprint einer Klasse zu repräsentieren. Die jeweilige Länge der Methodennamen lassen sich allerdings sehr wohl für eine Differenzierung innerhalb eines fingerprintbasierten Screenings nutzen. Dazu werden einfach die Längen der jeweiligen Methodennamen in den Fingerprint der zugehörigen Klasse codiert, so dass sich für jede Klasse ein spezifisches Muster ergibt.

```
Abbildung der lexikalischen Eigenschaft getTemperature

Abbildung Länge         000000 000000 010000 000000 00
Abbildung Buchstaben    100010 100000 100101 011000 00
Abbildung Wortteile     001000 000100 000000 000000 00
```

Abbildung 21: Abbildung lexikalische Eigenschaften

Abbildung der Buchstaben einer Zeichenkette

Eine weitere Möglichkeit für die Abbildung einer lexikalischen Eigenschaft ist, 26 Bitstellen in einem Fingerprint zu reservieren. Jede Bitposition entspricht dabei einem Buchstaben des Alphabets. Kommt der Buchstabe in dem Namen des Elementes vor, wird die entsprechende Bitposition gesetzt. Da bei diesem Ansatz ähnliche Zeichenketten mit einem ähnlichen Bitmuster repräsentiert werden, ist leicht ersichtlich, dass diese Form der Darstellung sich effektiv für ein fingerprintbasiertes Screening nutzen lässt.

Nachteil dieser Abbildung ist allerdings, dass sie sowohl von der Reihenfolge, als auch von der Häufigkeit in der die Buchstaben vorkommen, abstrahiert. Weiterhin ist die Problematik zu bedenken, dass eine auf diese Weise abgebildete lexikalische Eigenschaft in der Regel mehrere Bitstellen in einem Fingerprint belegt und somit bei einer Berechnung der Ähnlichkeit mit Hilfe der vorgestellten Metriken auch stärker ins Gewicht fällt. Bei Elementen, die neben dem Eigennamen nur wenige abbildbaren Merkmale aufweisen, kann dies dazu führen, dass die Berechnung ihrer Ähnlichkeit somit fast ausschließlich auf deren Namen basiert.

Abbildung einer Zeichenkette als einfaches Merkmal

Treude schlägt in [Tre07] auf Seite 43 einen weiteren Ansatz für die Abbildung von Elementnamen auf die Indizes eines Vektors vor. Dabei wird in einem ersten Schritt untersucht, welche Namen in den zu vergleichenden Dokumenten vorkommen. Im Folgenden wird jeder gefundene Namen auf eine bestimmte Position innerhalb des Vektors abgebildet. Elemente, die den gleichen Namen tragen, enthalten somit auch an identischen Vektorpositionen den Wert *1*. Der Ansatz auf Basis ganzer Namen ermöglicht natürlich nur eine sehr grobe Abbildung lexikalischer Eigenschaften. Deshalb wird ferner vorgeschlagen auch Teile der gefundenen Namen als Indizes abzubilden. Vergleiche hierzu auch Abbildung 21 *Abbildung Wortteile,* wobei die dritte Bitposition *get* repräsentiert und die zehnte Position für *Temperature* steht. Die abzubildenden Wortteile sind bei diesem Ansatz entweder über eine sinnvolle Definition von Trennstellen, beispielsweise anhand gängiger Konventionen innerhalb der Domäne, oder mittels eines Algorithmus wie LCS zu bestimmen. Ein solcher Ansatz ist natürlich auch bei der Nutzung eines Fingerprintverfahrens denkbar, wobei die Einschränkung gilt, dass der zusätzlich durch eine Analyse entstehende Aufwand der Zielsetzung eines fingerprintbasierten Screenings widerspricht.

Es wurde in *Kapitel 4.2 Unterschiede der Einsatzgebiete* gezeigt, dass sich im Gegensatz zu dem ursprünglichen Einsatzgebiet eines fingerprintbasierten Screenings die Ähnlichkeit von Elementen im Kontext einer Dokumentdifferenzierung in der Regel nicht an charakteristischen Strukturen, sondern vielmehr an lexikalischen und metrischen Elementeigenschaften orientiert. Dazu sind in diesem Kapitel verschiedene Konzepte für eine Abbildung derartige Merkmale besprochen, sowie deren Stärken und Schwächen aufgezeigt worden. Wie dabei gezeigt wurde, sind nicht alle denkbaren Abbildungen sinnvoll und einige lassen sich nur unter bestimmten Bedingungen für eine effektive Abbildung charakteristischer Merkmale nutzen.

5. Evaluation

5.1 Einleitung

Zu Beginn dieser Arbeit wurde gezeigt, wie ein fingerprintbasiertes Screening bei Suchanfragen auf molekularen Datenbanken zum Einsatz kommt. Seine Aufgabe dabei ist es, schnell eine Menge von aussichtsreichen Kandidaten für die jeweilige Suchanfrage aus der Datenbank zu extrahieren. Im Folgenden sind zwei verschiedene Ansätze für die Bildung von Fingerprints vorgestellt worden. Weiterhin wurde der von *Wehren* ([Weh04]) vorgeschlagene Algorithmus zur Differenzanalyse von technischen Dokumenten kurz skizziert. Dabei konnte trotz deutlicher Unterschiede der gegebenen Rahmenbedingungen aufgezeigt werden, dass der Ansatz eines Screenings auch in dem Kontext einer Differenzanalyse von Interesse ist. Auf Basis einer Implementierung des von *Wehren* vorgeschlagenen Algorithmus (*SiDiff*) wurde die Implementierung eines an die geänderten Rahmenbedingungen angepassten Verfahrens für ein Screening vorgestellt. Zielsetzung dabei war, anhand einer schnellen Analyse der Ähnlichkeit auf Basis von Fingerprints zu entscheiden, ob zwei Elemente als aussichtsreiche Kandidaten für eine hohe Ähnlichkeit in Frage kommen. Ist dies der Fall, so sind die Elemente mit weiteren Vergleichsfunktionen zu untersuchen. Die mit dem Einsatz des Verfahrens angestrebte Verbesserung der Laufzeit basiert einzig auf Einsparungen in der Berechnung kostenintensiver Vergleichsfunktionen.

Dieses Kapitel beschäftigt sich eingehend mit der Evaluation der vorgeschlagenen Implementierung des Fingerprintverfahrens. Dabei werden insbesondere Eigenschaften von Elementtypen, die aus dem jeweils zugrundeliegenden Modellspezifikationen resultieren, näher betrachtet. Für die Evaluation des Fingerprintverfahren ist als Vergleichsmaßstab der Einsatz von SiDiff auf Basis des paarweisen Elementvergleichs festgelegt und im Weiteren unter dem Begriff *SiDiff* zu verstehen. Stellen, wo als Ergänzung weitere Verfahren zum Einsatz kommen, sind explizit gekennzeichnet.

Es werden drei verschiedene Domänen untersucht. Aus der *Unified Modeling Language* (kurz: *UML*) werden *UML-Klassendiagramme,* sowie *UML-Aktivitätsdiagramme* auf ihre Eignung für den Einsatz eines Fingerprintverfahrens

geprüft. Weiterhin werden *MATLAB/Simulink-Diagramme*, die bei der Modellierung, Simulation sowie Analyse von dynamischen Systemen zum Einsatz kommen, auf ihre Eignung für ein fingerprintbasiertes Screening untersucht.

Bevor in den einzelnen Abschnitten die verschiedenen Ansätze der jeweils eingesetzten Fingerprints und die erzielten Ergebnisse näher beschrieben werden, erfolgt jeweils eine kurze Einführung in die vorliegende Domäne.

Die Evaluation ist auf vier Rechnern in Raum H-C 8327 des Hölderlingebäudes der Universität Siegen durchgeführt worden. Alle im Folgenden vorgestellten Werte sind, soweit nicht extra gekennzeichnet, als aggregierte Ergebnisse der durchgeführten Testläufe zu verstehen.

5.2 Unified Modeling Language

5.2.1 Einführung UML

Zwischen 1985 und 1995 wurden in dem Anwendungsgebiet der Softwareentwicklung eine Vielzahl [23] verschiedener Modellierungssprachen und Modellierungsansätze vorgeschlagen. Alle Ansätze sind dabei letztendlich vor einem identischen Forschungshintergrund propagiert worden. Der Prozess der Softwareentwicklung sollte vereinfacht und standardisiert werden. Dadurch sollte eine bessere Kommunikation, sowohl projektintern als projektextern, ermöglicht werden und die weitere Entwicklung des Anwendungsgebietes Softwaretechnik auf Grundlage einer einheitlichen Basis erfolgen. Wie allerdings bei einer derartigen Vielzahl von Ansätzen abzusehen war, endeten die Standardisierungsversuche in konkurrierenden Modellierungssprachen, die zum Teil gravierende Unterschiede aufwiesen, sowohl in ihren Ansätzen und Zielen als auch in der gewählten Darstellung der modellierten Systeme. Die eigentliche Intention der Ansätze ist somit nicht nur verfehlt worden, sondern die Kommunikation über Software und Softwaresysteme wurde erschwert.

Mit *Booch*, *Rumbaugh* und *Jacobson* unternahmen drei der einflussreichsten Sprachschöpfer ab ca. 1995 den Versuch einer Vereinheitlichung der unterschiedlichen Ansätze. Durch ihre Bemühungen entstand 1997 der erste Vorläufer der heutigen *Unified Modeling Language*. Seit ihrer Vorstellung ist die UML einer kontinuierlichen Weiterentwicklung unterzogen worden und kann heute als der Standard in der Softwaretechnik angesehen werden. Um eine zukünftige, von einer breiten Masse getragene, Weiterentwicklung sicherzustellen, übernahm die *Object Management Group* (*OMG*) [24] die Schirmherrschaft über den Standard. Die *OMG* ist ein 1989 ins Leben gerufenes Konsortium unterschiedlichster Marktteilnehmer aus Wirtschaft und Forschung. Ihr erklärtes Ziel ist die Entwicklung von herstellerunabhängigen und systemübergreifenden Standards für die objektorientierte Programmierung.

[23] *Für eine Übersicht der Ansätze sei auf ([Rup05], S. 13* Abbildung 1.1 Auf der Suche nach der „Lösung“ – Der Methodenkrieg *verwiesen*

[24] *http://www.omg.org/*

In der heute vorliegenden *Version 2* bietet die UML Notationen für unterschiedliche Aspekte der Modellierung von Softwaresystemen. Insbesondere die Bereiche Dokumentation, Spezifikation und Visualisierung komplexer Systeme werden unterstützt. Dabei ist hervorzuheben, dass innerhalb der UML von vorliegenden Fach- und Realisierungsgebieten, sowie von implementierungsspezifischen Eigenschaften - beispielsweise der Wahl einer Programmiersprache - abstrahiert wird.

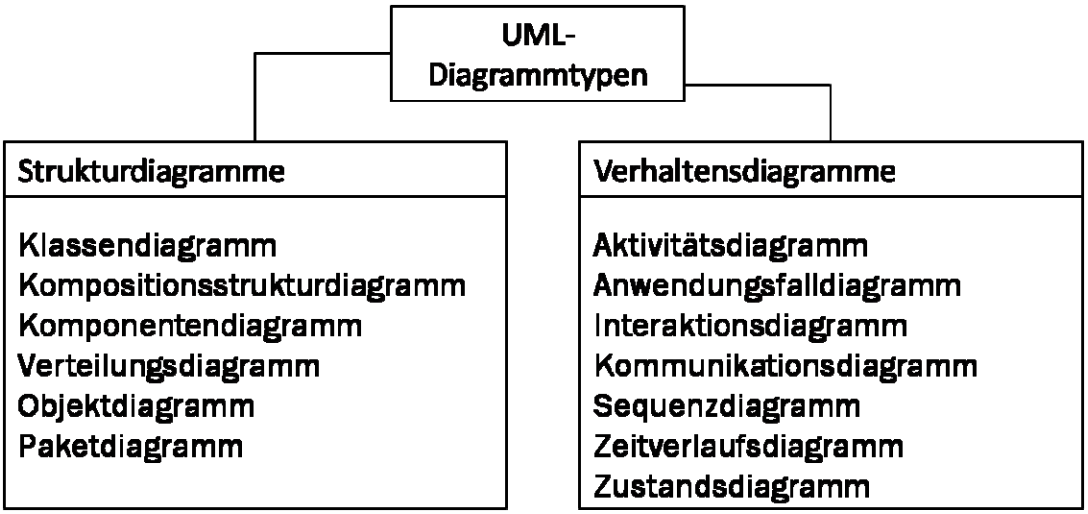


Abbildung 22: Übersicht UML-Diagrammtypen

In der UML kann zwischen 13 Diagrammtypen (siehe Abbildung 22) unterschieden werden, wobei eine Unterteilung zwischen Struktur- und Verhaltensdiagrammen erfolgt.

5.2.2 Aktivitätsdiagramme

Aktivitätsdiagramme (engl. *activity diagrams*) sind innerhalb der UML den Ablaufdiagrammen zuzuordnen. Ihr Haupteinsatzgebiet ist die Modellierung von Arbeitsabläufen, wobei sie sich insbesondere für die Spezifikation von Geschäftsprozessen (engl. *workflows*) und komplexen Vorgängen (vgl. [See00], S. 27) eignen. Im Rahmen der Softwareentwicklung wird dieser Diagrammtyp innerhalb von Analyseprozessen für die Abbildung dynamischer Anforderungen des zu modellierenden Systems eingesetzt.

Ein Aktivitätsdiagramm ist in Form eines Graphs (vgl. [For07], S. 145) repräsentiert. Knoten stehen dabei für Aktivitäten, Aktionen, Objekte sowie für weitere Kontrollelemente der Ablaufsteuerung. Die Kanten hingegen stellen Steuer- und Kontrollflüsse dar.

Ein Aktivitätsdiagramm besteht aus einem Startknoten und mindestens einem Endknoten. Zwischen dem Start- und Endpunkt werden die Aktivitäten definiert, die im Rahmen des modellierten Prozesses durchzuführen sind. Dabei stehen weitere Kontrollelemente zur Verfügung, welche beispielsweise Möglichkeiten bieten die Abläufe zu parallelisieren, Verzweigung des Ablaufs anhand von Entscheidungen vorzunehmen oder nebenläufige Aktivitätsstränge wieder zusammenzuführen. Weiterhin ist für Aktionen eine Möglichkeit gegeben, Verantwortungsbereiche (siehe [Hit03], S. 170) zu definieren und so festzulegen, welches Objekt für die Durchführung zuständig ist.

Auch wenn Aktivitätsdiagramme alle Möglichkeiten bieten äußerst komplexe Prozesse zu beschreiben, so werden sie doch hauptsächlich in der Praxis zur Abbildung einfacher Sachverhalte, beziehungsweise von Sachverhalten auf einer hohen Abstraktionsebene, genutzt.

Evaluationsansatz für Aktivitätsdiagramme

Auf Grund der im letzten Absatz genannten primären Nutzung von Aktivitätsdiagrammen in der Praxis war es nicht möglich reale Testdaten für diese Domäne zu finden. Gefundene Aktivitätsdiagramme wiesen typischerweise nur 20 bis 50 Elemente auf. Ihnen fehlte somit eine ausreichende Größe, um auf ihrer Basis aussagekräftige Ergebnisse für die Bewertung des Einsatzes eines fingerprintbasierten Screenings zu ermitteln. Weiterhin war es auf Grund von proprietären Datenformaten auch nicht möglich, gefundene Dokumente geeignet zu verändern und an die vorliegenden Anforderungen anzupassen.

Wegen den beschriebenen Schwierigkeiten sind die nötigen Testdaten für die folgende Evaluierung eines fingerprintbasierten Screenings selbst entwickelt worden. Mithilfe des UML-Editors *Poseidon*[25] wurden vier Dokumente entworfen, welche inhaltlich verschiedene Versionen eines Tagesablaufs modellieren. Jedes der Aktivitätsdiagramme enthält dabei etwa 250 bis 300 Elemente.

Untersucht man die vorliegende Domäne genauer, so sind die Elementtypen Aktivitätskante (*activityEdge),* Aktion *(callAction)* und Kontrollknoten *(controlNode)* als die zentralen Elemente für die Darstellung eines Aktivitätsdiagramms auszumachen. Diese Elementtypen spezifizieren sowohl alle durchzuführenden Aktionen, als auch alle Kontrollentscheidungen, die bei der Durchführung eines abgebildeten Prozesses zu berücksichtigen sind. Sie spielen auch in ihrer Anzahl die klar dominierende Rolle in Aktivitätsdiagrammen und weisen die größten Kandidatenmengen (vgl. Tabelle 3, *SiDiff*) im Rahmen eines paarweisen Vergleichs auf.

Betrachtet man die modellspezifischen Eigenschaften der dominierenden Elementtypen, so fällt auf, dass sie nur wenige Möglichkeiten für eine sinnvolle Differenzierung anhand allgemeiner und struktureller Merkmale aufweisen. Aktionen besitzen in der Regel jeweils eine Verbindung zu einer eingehenden, sowie eine Verbindung zu einer ausgehenden Aktivitätskante. Dabei ist zu bedenken, dass diese Eigenschaft nicht zwingend vorliegen muss und sie sich im Laufe der Entwicklung eines Aktivitätsdiagramms durchaus auch ändern kann, beispielsweise wenn bei einem spezifizierten Ablauf anhand einer zusätzlichen Überprüfung auf Abgeschlossenheit gegebenenfalls ein Rücksprung innerhalb des Diagramms erfolgt (vgl.[For07], S. 146f, Abbildung 2.129 u. 2.132). Es erscheint aus diesem Grund wenig sinnvoll, ein derartiges strukturelles Kriterium einzig als Basis für ein Screening anzulegen.

Es ist weiter festzustellen, dass insbesondere für Aktionen modellimmanent von einer aussagekräftigen Namensgebung ausgegangen wird. Dies spiegelt sich auch in

[25] *Homepage des Herstellers: http://www.gentleware.com/*

der SiDiff zugrundeliegenden Konfiguration für die Berechnung der Ähnlichkeit zwischen diesen Elementen (siehe *Anhang B I: umlActivityCompareConfig.xml (Auszug)*) wider. Dort ist zu erkennen, dass die Ähnlichkeit zwischen Aktionen einzig anhand der Ähnlichkeit ihres Namens ermittelt wird. Auch bei Kontrollelementen sind die beiden lexikalischen Eigenschaften *name* und *kind* die entscheidenden Faktoren bei der Bestimmung des Ähnlichkeitswertes. Für Aktivitätskanten ist allerdings zu sehen, dass sich die Berechnung der Ähnlichkeit hauptsächlich darauf stützt, ob zwischen den Elementen aus der Nachbarschaft eine Korrespondenz beziehungsweise eine hohe Ähnlichkeit festzustellen ist. Bedenkt man allerdings, dass benachbarte Elemente von Aktivitätskanten insbesondere Aktionen und Kontrollknoten sind, deren Ähnlichkeitsberechnung sich wieder auf ihre lexikalischen Eigenschaften stützt, so ist es in diesem Fall möglich, die lexikalischen Eigenschaften der benachbarten Elemente im Rahmen eines Fingerprints für Aktivitätskanten zu nutzen.

Es wurde eingangs festgestellt, dass Aktivitätsdiagramme insgesamt eine relativ geringe Anzahl an Elementen aufweisen und weiterhin die Berechnung der Ähnlichkeiten auf Basis weniger, hauptsächlich lexikalisch ausgerichteter Vergleichsfunktionen erfolgt. Somit erfüllen Dokumente der Domäne einige der als vorteilhaft für den Einsatz eines Fingerprintverfahrens angesehenen Eigenschaften nicht. Sehr wohl weisen die dominierenden Elementtypen allerdings distinkte Merkmale auf, die sich im Rahmen eines fingerprintbasierten Screenings nutzen lassen. Insgesamt ist abzuwarten, ob der Einsatz eines Fingerprintverfahrens gegenüber einem paarweisen Vergleich der Elemente einen Vorteil bringt.

Im Rahmen der Evaluierung eines fingerprintbasierten Screening für UML-Aktivitätsdiagramme werden aus den genannten Gründen zwei unterschiedliche Ansätze für die Bildung eines Fingerprints vorgestellt und genauer untersucht:

- Ein Fingerprint basierend auf strukturellen bzw. allgemeinen Charakteristika
- Ein Fingerprint basierend auf lexikalischen Eigenschaften der Elemente

Zwei Elemente werden bei den beiden Ansätzen als grundsätzlich ähnlich angesehen, wenn sie mindestens einen Tanimotokoeffizienten von $Tan = 0,7$ aufweisen. Für den lexikalischen Fingerprint wird zudem ein komplexer Ansatz zur Bildung der Kandidatenmenge vorgeschlagen, welcher im Folgenden noch näher erläutert wird.

Fingerprint über strukturelle und allgemeine Eigenschaften

In diesem ersten Ansatz kommt ein Fingerprint (siehe *Anhang A II: Fingerprint über strukturelle und allgemeine Eigenschaften*) zum Einsatz, der auf allgemeinen Eigenschaften der in dem internen Datenmodell repräsentierten Elemente basiert. Dabei wird hauptsächlich die Vernetzung der Elemente innerhalb der Graphrepräsentation auf Basis ein- und ausgehender Kanten, sowie weitere allgemeine Eigenschaften der Elemente und ihrer direkten Nachbarschaft abgebildet. Insofern ist für die Nutzung des vorgeschlagenen Fingerprints kein Wissen aus der jeweiligen Domäne nötig und er ist generisch für alle Modelle, die sich in dem in Abbildung 12 vorgeschlagenen internen Datenmodell repräsentieren lassen. Der hier vorgestellte Ansatz kann also auch für die Evaluierung weiterer Domänen eingesetzt werden.

Fingerprint über lexikalische Eigenschaften

Bei diesem Ansatz werden die Fingerprints für Aktionen anhand ihres Namens erstellt. Für Kontrollknoten wird neben dem Namen - sofern einer vergeben wurde - auch das Attribut *kind*, das festlegt um welchen Typ von Kontrollknoten es sich handelt, als lexikalische Eigenschaft aufgefasst und in dem Fingerprint abgebildet. Weiterhin werden für Aktivitätskanten insbesondere die Namen der beiden benachbarten Elemente innerhalb des Fingerprints repräsentiert.

Fingerprint über lexikalische Eigenschaften - Spezial

Bei dem letzten vorgeschlagenen Evaluationsansatz für diese Domäne kommt der schon im letzten Absatz vorgestellte Fingerprint erneut zum Einsatz. Allerdings wird diesmal nicht die Ähnlichkeit über die gesamten Fingerprints, sondern es wird die Ähnlichkeit über einzelne markante Teile der Fingerprints untersucht. Aktivitätskanten beispielsweise werden schon als grundsätzlich ähnlich angesehen, wenn sie eine hohe Ähnlichkeit in dem Namen eines der beiden Nachbarelemente aufweisen. Auf diese Weise wird verhindert, dass unter Umständen eine einseitige Änderung der Nachbarschaft auftritt, welche die Ähnlichkeitsberechnung über die gesamte Länge des Fingerprints verzerrt und zu einer Filterung eines aussichtsreichen Kandidaten führt. Weiterhin liegt der Schwellenwert für diesen Ansatz bei einem Tanimotokoeffizienten von $Tan = 0{,}6$.

Ergebnisse der Evaluation für Aktivitätsdiagramme

In Tabelle 2 sind die Ergebnisse der Testläufe dargestellt. Die Spalte *Korrespondenzen* gibt an, wie viele Zuordnungen zwischen Elementen bei allen Testläufen über alle Dokumente gefunden wurden. Die Spalte *Zeit* spiegelt die für die Differenzanalyse benötigte Zeit wider. Um einen einfacheren Vergleich der Ergebnisse zu ermöglichen, sind einzig die absoluten Werte für den paarweisen Vergleich von SiDiff angegeben und für die verschiedenen Ansätze eines Screenings die jeweils dazu aufgetretene prozentuale Veränderung. Eine negative Prozentzahl in der Spalte *Zeit* steht demnach für eine Verbesserung der Laufzeit, während eine negative Prozentzahl in der Spalte *Korrespondenzen* für weniger gefundene Zuordnungen steht. In Tabelle 3 sind die festgestellten Veränderungen in den Kandidatenmengen aufgeführt. Die Mengen wurden bestimmt, indem vor Beginn der Ähnlichkeitsberechnung über die Elemente aus dem zuerst für den Vergleich festgelegten Dokument iteriert, die Anzahl der Kandidaten bestimmt und aufgeschlüsselt nach Elementtyp aufaddiert wurden. Auch hierbei sind wieder für eine verbesserte Vergleichbarkeit nur die von SiDiff ermittelten Werte als absolute Werte angegeben und die jeweils durch ein Screening erzielten Veränderungen in Relation dazu. Steht in einem Feld *0%*, so trat keine Veränderung zu der von SiDiff angenommenen Kandidatenmenge auf.

	SiDiff	***Struktur***	***Lexikalisch***	***Spezial***
Zeit	57.6sek	4.0%	-13.4%	-12.3%
Korrespondenzen	7064	-0.1%	-4.2%	0.0%

Tabelle 2: Übersicht Ergebnisse Aktivitätsdiagramme

Fingerprint über strukturelle und allgemeine Eigenschaften

Wie in Tabelle 2 zu sehen ist, führt ein Fingerprint auf Basis struktureller und weiterer allgemeiner Merkmale in etwa zu der gleichen Anzahl von Zuordnungen wie der reguläre Einsatz von SiDiff. Dies war allerdings nicht anders zu erwarten, da schon gezeigt wurde, dass die strukturellen Eigenschaften der einzelnen Elementtypen in dieser Domäne ausgesprochen starr sind und sich auf ihrer Basis dementsprechend keine sinnvolle Reduktion der Kandidatenmengen (vgl. Tabelle 3) durchführen lässt. Untersucht man die gefundenen Zuordnungen genauer, so ist festzustellen, dass der Rückgang an gefundenen Korrespondenzen durch

Filtereffekte [26] bei Kontrollknoten, welche in verschiedenen Versionen des Dokumentes Veränderungen in der Anzahl der ein- bzw. ausgehenden Aktivitätsstränge aufweisen, zu erklären ist.

Die Durchführung der Differenzbildung zwischen den Dokumenten unter Einsatz dieses Fingerprints dauert auch minimal länger als eine Durchführung ohne Einsatz eines strukturbasierten Screenings.

	SiDiff	*Struktur*	*Lexikalisch*	*Spezial*
model	48	0.0%	0.0%	0.0%
collaboration	48	0.0%	0.0%	0.0%
controlNode	25376	-5.6%	-89.4%	-71.8%
callAction	243424	0.0%	-96.4%	-74.7%
activityEdge	559848	-4.3%	-99.0%	-90.6%
activity	48	0.0%	0.0%	0.0%

Tabelle 3: Veränderung Kandidatenmengen Aktivitätsdiagramme

Fingerprint auf Basis lexikalischer Merkmale

Der Einsatz eines fingerprintbasierten Screenings auf Basis lexikalischer Merkmale führt zu einer mitunter deutlichen Verringerung der Kandidatenmengen (siehe Tabelle 3) und dementsprechend auch zu einer deutlichen Verbesserung der Laufzeit um etwa 13%. Allerdings werden bei einem Screening auf dieser Basis auch etwas weniger Korrespondenzen ermittelt. Grund hierfür ist, dass die Bildung einer Kandidatenmenge auf Basis des gesamten Fingerprints insbesondere für Aktivitätskanten bei einer einseitigen Veränderung der Nachbarschaft zu der bereits erwähnten Filterung von aussichtreichen Kandidaten führen kann. Es ist festzustellen, dass der starke Rückgang in den gefundenen Korrespondenzen auf den beschriebenen Effekt zurückzuführen ist.

[26] *Im Folgenden ist unter dem Begriff eines* Filtereffekts *jeder Unterschied in den gefundenen Korrespondenzen zu verstehen, der sich auf die Bildung einer Kandidatenmenge zurückführen lässt. In dem vorliegenden Fall führt der eingetretene Filtereffekt einzig zu weniger erkannten Korrespondenzen. Wie später bei der Evaluation von Klassendiagrammen noch gezeigt wird, können auch weitere Arten von Filtereffekten auftreten.*

Fingerprint auf Basis lexikalischer Merkmale - Spezial

Erfolgt auf Basis des gleichen Fingerprints allerdings eine differenzierte Bildung der Kandidatenmengen, so lässt sich ein gutes Ergebnis erzielen. Die Durchführung der Differenzanalyse erfolgt mit diesem Screening 12% schneller und es werden gleich viele Zuordnungen gefunden.

Fazit

Abschließend bleibt für den Einsatz eines fingerprintbasierten Screenings in dieser Domäne festzuhalten, dass sich das Verfahren bewährt hat. Die Laufzeit der Dokumentdifferenzierung konnte um etwa 12% verbessert werden. Dies zeigt, dass auch bei Ähnlichkeitswerten, die sich nur aus wenigen Kriterien zusammensetzen, eine Verbesserung der realen Laufzeit mithilfe eines fingerprintbasierten Screenings erzielt werden kann.

Weiterhin ist aus dem erzielten Ergebnis zu schließen, dass die Vergabe eines aussagekräftigen Namens für einen Elementtyp als Selektionskriterium für die Entscheidung ausreicht, ob eine genaue Berechnung der Ähnlichkeit zwischen Elementen durchzuführen ist oder nicht.

Insgesamt ist allerdings abzuwägen, ob sich der Einsatz eines fingerprintbasierten Screenings in dieser Domäne lohnt. Zum einen ist fraglich, ob reale Dokumente eine ausreichende Anzahl an Elementen aufweisen. Weiterhin ist die durchgeführte Differenzanalyse für diesen Modelltyp aufgrund der wenigen zu berechnenden Ähnlichkeitskriterien als grundsätzlich schnell anzusehen, so dass die erzielten 12% Einsparung an Laufzeit nur wenige Sekunden ausmacht.

5.2.3 Klassendiagramme

Klassendiagramme sind innerhalb der UML den Strukturdiagrammen zuzuordnen. Ein Klassendiagramm fungiert dabei als eine statische Repräsentation eines Softwaresystems und eine abgebildete Klasse beschreibt eine Menge an Objekten, die sich in ihren Eigenschaften, Einschränkungen und ihrer Struktur (vgl. [Rup05], S. 96) gleichen. Neben Eigenschaften und Verhalten beschreibt ein Klassendiagramm weiterhin, welche Beziehungen zwischen Klassen existieren. Klassendiagramme nutzen, ebenso wie die bereits vorgestellten Aktivitätsdiagramme, eine überschaubare Menge an Elementtypen für die Modellierung eines Systems.

Für Softwaresysteme sind eine Reihe von Ansätzen vorgeschlagen worden, wie sich deren Eigenschaften auf Zahlen abbilden lassen. Entwicklungs- und Einsatzgebiet dieser metrischen Indizes, auch *Softwaremetriken* genannt, ist der Bereich des Softwarequalitätsmanagement. Einige[27] der vorgeschlagenen Metriken für Klassenelemente (in Anlehnung an [Lan99], S. 11) sind in Tabelle 4 abgebildet.

Name	***Beschreibung***
NCV	*Number of class variables;* Anzahl der Klassenvariablen
NA	*Number of accessors*; Anzahl der get- und set-Methoden
NOM	*Number of methods;* Anzahl der Methoden
NOA	*Number of attributes;* Anzahl der Attribute
PriM	*Number of private Methods;* Anzahl der private-Methoden
ProM	*Number of protected Methods;* Anzahl der protected-Methoden
PubM	*Number of public Methods;* Anzahl der public-Methoden
WLOC	*Lines of Code;* Anzahl der Codezeilen

Tabelle 4: Ausgewählte Softwaremetriken

Wie ersichtlich ist, lassen sich nicht alle der vorgeschlagenen Metriken aus UML-Klassendiagrammen ableiten. Beispielsweise wird für die Ermittlung der Metrik *WLOC - Lines of Code* Kenntnis des Quelltexts benötigt. Eine derartige Metrik ist für den gegeben Anwendungsfall irrelevant und kann im Weiteren vernachlässigt werden. Weiterhin ist festzustellen, dass die vorgeschlagenen Metriken sich meist auf einfache Zähloperationen (vgl. [Tre07], S.41) zurückführen lassen, entweder auf Basis des Typs benachbarter Elemente (vgl. *NCV – Number of class variables*) oder auf Basis benachbarter Elemente eines Typs, die zudem einen bestimmten Attributwert aufweisen (vgl. *PriM – Number of private methods*).

[27] *Für eine ausführlichere Übersicht und Metriken für weitere Elemente sei auf [Lan99] verwiesen.*

Alle vorgestellten Softwaremetriken lassen sich - sofern sie in der internen Modellierung eines Klassendiagramms präsent sind - in einem Fingerprint abbilden und somit für ein fingerprintbasiertes Screening nutzen.

Dabei ist allerdings zu beachten, dass bei Nutzung von Softwaremetriken mitunter Abhängigkeiten unter den einzelnen Metriken bestehen. Die Änderung einer Eigenschaft kann sich simultan in mehreren Softwaremetriken niederschlagen und deren Wert verändern. Ein Beispiel für diesen Effekt ist das Hinzufügen einer *Get-Methode* mit der Sichtbarkeit *public*. Neben einer Änderung des Wertes der Softwaremetrik *PubM - number of public methods* hat ein Hinzufügen auch Auswirkungen auf den Wert der Metrik *NOM - number of methods*, sowie der Metrik *NA – Number of accessors.* Werden mehrere Metriken, die von der gleichen Eigenschaft abhängig sind, in einem Fingerprint abgebildet, so ändern sich in einem solchen Fall entsprechend mehrere Bitstellen. Dieses Verhalten kann dazu führen, dass zwei Fingerprints selbst nach geringfügigen Änderungen kaum noch Ähnlichkeit zueinander aufweisen.

Evaluationsansatz UML-Klassendiagramme

Die Evaluation eines fingerprintbasierten Screenings für UML-Klassendiagramme erfolgt auf Basis von Dokumenten, die aus dem Quellcode des Projekts *Fujaba* [28] der Universität Paderborns mittels Re-Engineering erstellt wurden. Insgesamt stehen für den Test sieben verschiedene Klassendiagramme zur Verfügung. Bei den vorliegenden Testdaten handelt es sich um ausgesprochen umfangreiche Dokumente. Im Schnitt besteht ein Diagramm aus etwas über 8000 Elementen. Anders als bei den Aktivitätsdiagrammen sind Dokumente dieser Größenordnung für Klassendiagramme nicht ungewöhnlich, da in ihnen mitunter äußerst komplexe Systeme modelliert werden. Als dominierende Elementtypen innerhalb dieser Domäne können Methoden *(operation)*, Parameter *(parameter)*, Attribute *(attribute)* und Klassen *(class)* ausgemacht werden. Diese vier Elementtypen bilden den Kern für die Repräsentation von Klassen und ihren Eigenschaften. Dementsprechend groß ist auch die bei einem paarweisen Vergleich zu betrachtenden Kandidatenmengen (siehe Tabelle 6).

Allerdings sind UML-Klassendiagramme nicht nur durch die große Anzahl an Elementen der dominierenden Typen ein interessantes Einsatzgebiet für den vorgestellten Ansatz eines Screenings. Die Berechnung eines Ähnlichkeitswerts zwischen zwei Elementen erfolgt bei vielen Elementtypen dieser Domäne auf Berechnung und Aufsummierung mehrerer Einzelkriterien (siehe *Anhang B II: umlClassCompareConfig.xml (Auszug)*). Beispielsweise ist für eine Berechnung des Ähnlichkeitswerts zweier Klassen zuvor eine Berechnung von sieben einzelnen Ähnlichkeitsfunktionen nötig. UML-Klassendiagramme erfüllen somit viele der für ein fingerprintbasiertes Screening als vorteilhaft eingeschätzten Eigenschaften. Wenn sich durch ein Screening eine sinnvolle Reduktion der entsprechenden Kandidatenmengen erzielen lässt, so ist davon auszugehen, dass dies zu einer deutlichen Einsparung an Laufzeit führt.

Innerhalb von Klassendiagrammen treten allerdings die in *Kapitel 4.1 Ein Algorithmus zur Berechnung von Dokumentdifferenzen* bereits beschrieben *Teil-von*-Beziehungen auf. Klassen beispielsweise bestehen aus Methoden und Attributen (vgl. [Kel06], S. 3). Dies ist nicht nur bei der Berechnung der Ähnlichkeitswerte zu berücksichtigen, sondern es ist weiterhin zu bedenken, dass nicht oder anders vorgenommene Zuordnungen von Elementen dieser Domäne zu einer Veränderung in den Ähnlichkeitswerten – und somit in den Zuordnungen - von weiteren Elementen führen kann, die mitunter auch unterschiedliche Typen aufweisen. Der Einsatz eines Verfahrens zur Bildung von Kandidatenmengen kann dementsprechend in der vorliegenden Domäne zu umfangreicheren Filtereffekten führen, als dies beispielsweise bei Aktivitätsdiagrammen der Fall ist. Diese werden im Weiteren noch genauer untersucht.

[28] *Homepage des Projekts: http://wwwcs.uni-paderborn.de/cs/fujaba/*

Auch bei Klassendiagrammen stellt sich bei einer genaueren Betrachtung der modellspezifischen Eigenschaften der dominierenden Elementtypen heraus, dass innerhalb des Modells die Vergabe eines Namens vorgesehen ist. Betrachtet man die von SiDiff vorgegebene Gewichtung der einzelnen Ähnlichkeitskriterien, so ist ebenso festzustellen, dass die Eigennamen der Elemente bei einer Berechnung der Ähnlichkeiten die wichtigste Rolle spielen. Aus diesem Grund erscheint in dieser Domäne erneut die Durchführung eines Screenings anhand lexikalischer Merkmale als besonders interessant.

Als Ansätze für ein fingerprintbasiertes Screening werden vier verschiedene Herangehensweisen für die Erstellung eines Fingerprints untersucht.

Diese Ansätze sind im Einzelnen:

- Ein Fingerprint basierend auf strukturellen bzw. allgemeinen Merkmalen
- Ein Fingerprint basierend auf den vorgestellten Softwaremetriken
- Ein Fingerprint über lexikalische Eigenschaften
- Ein Fingerprint über spezielle Eigenschaften

Für die ersten drei Ansätze wird als Ähnlichkeitsmaß für ein Screening erneut der Tanimotokoeffizient genutzt. Als Schwellenwert für eine Ähnlichkeit ist ein Wert von *Tan = 0,7* festgesetzt. Für den letztgenannten Ansatz hingegen kommt ein komplexer Ansatz zur Bildung einer Kandidatenmenge zum Einsatz. Dessen Vorgehensweise bei der Bestimmung möglicher Kandidaten wird im Weiteren noch genauer erläutert.

Fingerprint über strukturelle und allgemeine Charakteristika

Bei diesem Test wird ein Screening auf Basis des bereits in *Kapitel 5.2.2 Aktivitätsdiagramme* vorgestellten Fingerprints (siehe auch *Anhang A II: Fingerprint über strukturelle und allgemeine Eigenschaften*) durchgeführt.

Fingerprint über metrische Indizes

Der zweite Fingerprint (siehe *Anhang A IV: Fingerprint Softwaremetriken (UML-Klassendiagramme)*) basiert auf den bereits vorgestellten Softwaremetriken und es werden dementsprechend einzig zählbare Eigenschaften abgebildet. Es besitzen nicht alle Elementtypen in Klassendiagrammen Eigenschaften, die sowohl zählbar als auch sinnvoll in einen Fingerprint abzubilden sind. Aus diesem Grund beschränkt sich der vorgestellte Ansatz auf Klassen. Wie bereits gezeigt, ist es ein weiteres Problem hierbei, dass einige der Softwaremetriken nicht in der Darstellung eines UML-Klassendiagramms enthalten sind. Weiterhin können Abhängigkeiten zwischen einzelnen Softwaremetriken auftreten.

Fingerprint über lexikalische Indizes

Dieser Ansatz für einen Fingerprint stützt sich ausschließlich auf lexikalische Eigenschaften von Elementen. Neben den Eigennamen sind für ausgewählte Elementtypen – sofern dies sinnvoll erschien - auch lexikalische Eigenschaften von benachbarten Elementen in ihrem Fingerprint abgebildet. Für einen detailierten Überblick über die betrachteten Elementtypen und abgebildeten Eigenschaften siehe *Anhang A V: Fingerprint lexikalische Merkmale (UML- Klassendiagramme).*

Fingerprint über spezielle Eigenschaften

Der letzte Fingerprint (vgl. *Anhang A VI: Fingerprint spezielle Eigenschaften (UML-Klassendiagramme)*) der für eine Evaluierung vorgeschlagen wird, weist die höchste Komplexität der vorgestellten Ansätze auf. Für alle relevanten Elementtypen werden dabei einzig spezifische Eigenschaften in ihren Fingerprints abgebildet. Von diesen Merkmalen ist insbesondere zu erwarten, dass sie sich gut für die Bildung einer Kandidatenmenge nutzen lassen. Dabei wird im Weiteren nicht auf eines der vorgestellten Ähnlichkeitsmaße zurückgegriffen, sondern es erfolgt eine wesentlich differenziertere Bildung der Kandidatenmengen. Die Grundidee hierzu ist analog zu dem Ansatz für eine spezielle Kandidatenbildung, wie er bereits bei Aktivitätsdiagrammen erfolgreich zum Einsatz kam, zu sehen.

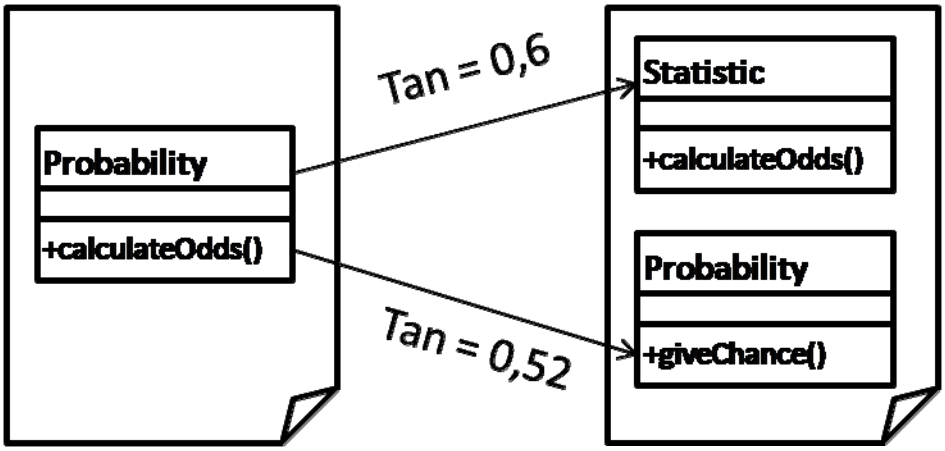


Abbildung 23: Beispiel Spezialfingerprint

Abbildung 23 zeigt für Klassendiagramme ein Beispiel einer Situation in der eine derart differenzierte Betrachtungsweise von Vorteil ist. Angenommen, es sind die dargestellten Methoden der drei Klassen aus den beiden Dokumenten zu vergleichen. In den Fingerprints einer Methode sind einzig der Methodenname, sowie der Name der zugehörigen Klasse überschneidungsfrei codiert. Es sei weiter angenommen, dass die Methode *Probability.calculateOdds()* einmal in die Klasse *Statistic* verschoben und einmal in *Probability.giveChance()* umbenannt wurde. Wird nun die Ähnlichkeit der Methoden über den gesamten Fingerprint auf Basis des Tanimotokoeffizienten ermittelt und gegen einen Schwellenwert von *Tan = 0,7* verglichen, so wird sowohl die Verschiebung, als auch die Umbenennung innerhalb des Screenings als nicht erfolgsversprechende Kandidaten angesehen und gefiltert. Wird ein Screening für Methoden allerdings einmal auf Basis des Namens der zugehörigen Klasse, sowie einmal auf Basis des Methodennamens durchgeführt, so ist leicht ersichtlich, dass sowohl eine Verschiebung wie auch eine Umbenennung nicht dazu führt, dass ein Element aus der Kandidatenmenge gefiltert wird. Werden in einem weiteren Teil des Fingerprints sonstige Eigenschaften von Methoden abgebildet, beispielsweise Informationen über übergebene Parameter oder den jeweiligen Rückgabetyp, kann mittels des vorgestellten Ansatzes eine weitere Differenzierung erfolgen. Das vorgestellte Konzept ist auf andere Elementtypen übertragbar, sofern diese geeignete Eigenschaften besitzen.

Dieser Ansatz eines fingerprintbasierten Screenings zielt auf eine weitaus weniger restriktive Filterung der Kandidatenmenge ab, als die bis hierher verfolgte Ähnlichkeitsbestimmung über ganze Fingerprints. Dadurch ist selbstverständlich mit

größeren Kandidatenmengen und daraus resultierend mit einer weniger hohen Einsparung an Laufzeit zu rechnen.

Ergebnisse der Evaluation für Klassendiagramme

Im Folgenden werden die Ergebnisse der Evaluierung für die Domäne der UML-Klassendiagramme vorgestellt. In Tabelle 5 ist eine Übersicht über die Anzahl der mittels der vorgestellten Ansätze gefundenen *Korrespondenzen* zu sehen, sowie die Zeit, die für die Berechnung der Differenzen benötigt wurde. Tabelle 6 stellt die bei den Ansätzen aufgetretenen Änderungen in der Anzahl der Kandidatenmengen dar. Wie schon bei der Evaluierung von Aktivitätsdiagrammen werden zwecks einer besseren Übersicht wieder nicht die absoluten Werte der Ergebnisse, sondern die jeweils eingetretene prozentuale Veränderung gegenüber den auf Basis von SiDiff ermittelten Vergleichswerten angegeben.

	Sidiff	***Struktur***	***Metrisch***	***Lexikalisch***	***Spezial***
Korrespondenzen	755827	-6.6%	-18.0%	1.8%	1.3%
Zeit	260min	-27.7%	15.0%	-81.5%	-31.2%

Tabelle 5: Übersicht Ergebnisse Klassendiagramme

Wie an den Ergebnissen zu erkennen ist, kommt es zwischen den verschiedenen vorgeschlagen Varianten eines fingerprintbasierten Screenings mitunter zu deutlichen Unterschieden. Dies lässt sich sowohl anhand der gemessenen Laufzeit der Differenzanalyse, als auch in der Anzahl der gefundenen Zuordnungen sehen. Im Folgenden werden die einzelnen Ergebnisse im Detail vorgestellt und interpretiert. Danach erfolgt ein weiterer, differenzierter Vergleich der mittels eines fingerprintbasierten Screenings gefundenen Zuordnungen zu den von SiDiff gefundenen Korrespondenzen.

Fingerprint über strukturelle und allgemeine Charakteristika

Ein Fingerprint, der nur auf Basis struktureller und allgemeiner Eigenschaften (siehe Tabelle 5, *Struktur*) eine Filterung der Kandidatenmengen vornimmt, führt zu einer spürbaren Verbesserung der Laufzeit und ist knapp 28% schneller als SiDiff. Allerdings ist auch zu erkennen, dass insgesamt etwas weniger Zuordnungen vorgenommen werden.

Bei einer genaueren Untersuchung der Gründe für den Rückgang der gefundenen Korrespondenzen stellte sich heraus, dass dieser Effekt insbesondere auf eine spezielle Eigenschaft von Klassen zurückzuführen ist. Wie beschrieben basiert der vorgeschlagene Ansatz hauptsächlich auf dem Vernetzungsgrad eines Elements innerhalb des Dokuments. Dieser kann sich für Klassen auf Basis zweier unterschiedlicher Effekte teilweise stark von Dokument zu Dokument verändern. Der erste Effekt entsteht durch das Hinzufügen und/oder Entfernen von Methoden und Attributen. Dies wirkt sich natürlich insbesondere auf die Anzahl der ausgehenden Kanten aus. Der zweite und wesentlich stärkere Effekt sind Veränderungen in der Nutzungshäufigkeit einer Klasse innerhalb des modellierten Systems, beispielsweise durch Nutzung als Parameter- oder Rückgabetyp von weiteren Methoden. Kommt es für eine Klasse in starkem Ausmaß zu einer der beschriebenen Veränderungen, so ist einfach ersichtlich, dass dies mitunter zu einer Filterungen führen kann, die als nicht optimal anzusehen ist.

Wie in Tabelle 6 zu sehen ist, lassen sich auch nicht alle Elementtypen anhand allgemeiner und struktureller Eigenschaften unterscheiden. So führt ein Einsatz auf Basis des Fingerprints beispielsweise für Klassenattribute zu keiner Reduktion der Kandidatenmenge. Da Attribute allerdings als einer der dominierenden Elementtypen dieser Domäne identifiziert wurden – und somit von einem hohen Interesse für ein Screening sind -, ist dies ein weiterer Kritikpunkt an dem vorgestellten Ansatz.

	SiDiff	Struktur	Metrisch	Lexikalisch	Spezial
operation	681697600	-57.9%	0.0%	-99.0%	-93.1%
model	168	0.0%	0.0%	0.0%	0.0%
parameter	314139472	-46.9%	0.0%	-99.1%	-81.0%
stereotype	672	0.0%	0.0%	0.0%	0.0%
association	5375080	0.0%	0.0%	-97.3%	0.0%
generalization	8445624	0.0%	0.0%	0.0%	0.0%
package	930584	-57.7%	0.0%	-98.1%	-98.0%
attribute	51370456	0.0%	0.0%	-98.5%	-62.9%
class	31055232	-83.5%	-66.7%	-98.7%	-79.3%
datatype	16320	-52.1%	0.0%	-81.8%	-81.8%
associationEnd	21500320	0.0%	0.0%	-85.0%	0.0%

Tabelle 6: Veränderung Kandidatenmengen Klassendiagramme

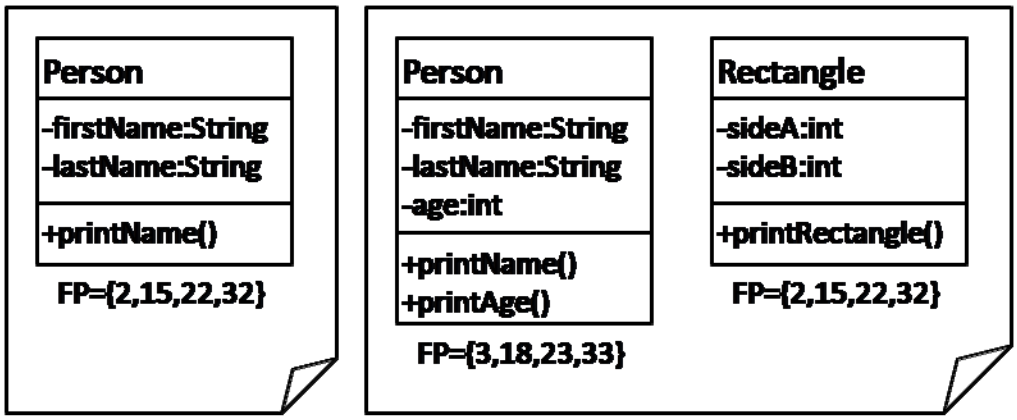
Es bleibt festzuhalten, dass ein Screening auf Basis allgemeiner und struktureller Eigenschaften zu einer spürbaren Verbesserung der Laufzeit führt. Weiterhin ist allerdings auch festgestellt worden, dass ein derartiger Ansatz weniger Zuordnungen findet. Die Gründe hierfür liegen in einer Reduktion der Kandidatenmenge, die mitunter als nicht optimal anzusehen ist. Insgesamt empfiehlt sich ein fingerprintbasiertes Screening einzig auf Basis dieses Ansatzes nur bedingt.

Fingerprintbasiertes Screening auf Basis metrischer Indizes

Die vorgestellten Softwaremetriken erwiesen sich bei der Evaluierung des Fingerprintverfahrens als nicht geeignet, um eine sinnvolle Reduktion der Kandidatenmengen durchzuführen. Dies ist vor allem auf zwei Gründe zurückzuführen:

Wie bereits dargestellt, beschränkt sich ein Screening bei diesem Ansatz einzig auf Klassen (vgl. Tabelle 6). Die dabei erreichte Reduktion der Kandidatenmenge fällt allerdings so gering aus, dass keine Verbesserung der Laufzeit eintritt. Im Gegenteil: Die Durchführung der Berechnung der Dokumentdifferenzen (vgl. Tabelle 5) ist im Vergleich zu SiDiff sogar deutlich langsamer. Der Aufwand für die Erstellung der Fingerprints ist zusammen mit dem Aufwand der anschließenden Filterung von Kandidaten demnach so groß, dass die erzielte Reduktion der Kandidatenmenge und die daraus resultierende Einsparung in der Anzahl an Vergleichsfunktionen nicht ausreicht, um ihn auszugleichen.

Das zweite Problem des Ansatzes liegt in der Tatsache, dass sich die abgebildeten Eigenschaften von Klassen über die verschiedenen Versionen der Dokumente mitunter deutlich ändern. Ein vereinfachtes Beispiel für diesen Sachverhalt ist in Abbildung 24 zu sehen. Hier ist sofort ersichtlich, dass bei einem Vergleich der beiden Dokumente die Klassen *Person* als potentielle Kandidaten für eine Korrespondenz in Frage kommen sollten. Bei einem Screening auf Basis eines Fingerprints[29] über die in der Abbildung zu sehenden Softwaremetriken kommt es zu einer Filterung. Durch das Hinzufügen eines einzelnen Attributs und einer weiteren Methode sind alle Metriken (es wird implizit von vorhandenen *get*- und *set*-Methoden für alle Attribute ausgegangen) verändert worden. Hingegen wird die Klasse *Rectangle*, da sie in Anzahl ihrer Attribute und Methoden mit der Klasse *Person* aus dem anderen Dokument übereinstimmt, sehr wohl als Kandidat für eine Zuordnung angesehen und im Weiteren genauer untersucht. Wie bereits bei der Interpretation der Ergebnisse eines strukturell-orientierten Screenings beschrieben, führt eine derartige Filterung mitunter zu nicht oder unterschiedlich vorgenommenen Zuordnungen.




```
---
<NumberOfPrivateAttributes mapping="0" />
<NumberOfMethods mapping="10" />
<NumberOfGetters mapping="20" />
<NumberOfSetters mapping="30" />
---
```

Abbildung 24: Beispiel Softwaremetriken

[29] *Die Fingerprints sind in den geschweiften Klammern zu sehen, wobei jede Zahl eine gesetzte Bitposition repräsentiert.*

Es überrascht weiterhin, dass der vorgestellte, speziell auf Klassen ausgelegte Ansatz eines fingerprintbasierten Screenings mittels Softwaremetriken eine deutlich größere Kandidatenmenge (siehe Tabelle 6) für Klassen findet, als der vorherigen Abschnitt vorgestellte Ansatz auf allgemeinen und strukturellen Elementeigenschaften. Bei einer genaueren Untersuchung des Sachverhalts hat sich gezeigt, dass in den vorliegenden Testdaten Klassen enthalten sind, die nur implizit in dem modellierten System vorkommen. Ein Beispiel hierfür ist eine Klasse, die einzig zur Spezifizierung des Rückgabewerts einer Methode aufgeführt ist. Derartige Klassen werden jedoch in der internen Darstellung als Klassenelemente repräsentiert, die einzig einen Namen und ansonsten keine weiteren Eigenschaften aufweisen. Somit bieten sie natürlich auch keinen Ansatz für eine Unterscheidung auf Basis der vorgestellten Softwaremetriken. Die Erstellung eines Fingerprints nach allgemeinen und strukturellen Merkmalen basiert hingegen auf der Vernetzung der Elemente innerhalb des Dokuments und bietet dementsprechend Möglichkeiten zur Unterscheidung der beschriebenen Klassen.

Wie gezeigt wurde führt der vorgestellte Ansatz eines Fingerprints auf Basis von Softwaremetriken weder zu einer sinnvollen Reduktion der Kandidatenmengen, noch zu Einsparungen bei der Laufzeit. Darüberhinaus ist festzustellen, dass auch dieser Ansatz zu weniger vorgenommenen Zuordnungen führt. Die Durchführung eines Screenings auf Basis von Softwaremetriken ist dementsprechend als nicht sinnvoll einzuschätzen.

Fingerprint über lexikalische Eigenschaften

Der Einsatz des Fingerprintverfahrens auf Basis lexikalischer Eigenschaften führt zu sehr guten Ergebnissen. Die Eigennamen von Elementen lassen sich auch in Klassendiagrammen äußerst effektiv für ein Screening verwenden. Dies schlägt sich letztendlich auch in der Laufzeit nieder und der Vergleich der Dokumente wird über 80% schneller (siehe Tabelle 5) durchgeführt als bei einem Einsatz von SiDiff ohne Screening. Außerdem konnte auf Basis des vorgestellten Screenings eine Steigerung in der Anzahl der gefundenen Zuordnungen von Elementen erreicht werden.

Dass bei diesem Ansatz mehr Zuordnungen vorgenommen werden, ist auf den ersten Blick ungewöhnlich. Bei näherer Betrachtung hat sich gezeigt, dass dieses Ergebnis letztendlich durch eine Auflösung auftretender Konflikte bei der Zuordnung von Elementen erreicht wird. Wie in *Kapitel 4.1 Ein Algorithmus zur Berechnung von Dokumentdifferenzen* beschrieben, findet eine Zuordnung von Elementen in SiDiff nur statt, wenn diese eindeutig erfolgen kann. Da sich ein Ähnlichkeitswert zweier Elemente in dieser Domäne in der Regel aus mehreren,

unterschiedlichen Einzelkriterien errechnet, tritt an einigen Stellen der in Abbildung 25 leicht vereinfacht dargestellte Fall ein.

Es wird in der Abbildung das Element *AccountManager* auf seine Ähnlichkeit zu den Elementen *AccountManagement* und *AssistantDirector* untersucht. Es sei angenommen, dass die Ähnlichkeitswerte der Elemente auf Basis einer komplexen Ähnlichkeitsberechnung mit *Sim = 0,76* identisch sind. Auf dieser Basis kann demnach keine Zuordnung erfolgen. Der jeweilige Fingerprint *FP* gibt an, welche Buchstaben in dem Namen der Elemente vorkommen. Wird ein fingerprintbasiertes Screening auf Basis der angegebenen Fingerprints durchgeführt und als Mindestkriterium für eine Ähnlichkeit der Tanimotokoeffizient mit einem Schwellenwert von *Tan = 0,7* festgelegt, kommt es zu einer Filterung des Elements *AssistantDirector* aus der Kandidatenmenge und dementsprechend kann eine Zuordnung der anderen beiden Elemente stattfinden. Eine solche zusätzlich erfolgte Zuordnung von Elementen kann sich, insbesondere wenn *Teil-von*-Beziehungen vorliegen, auch auf die Zuordnungen weiterer Elemente auswirken.

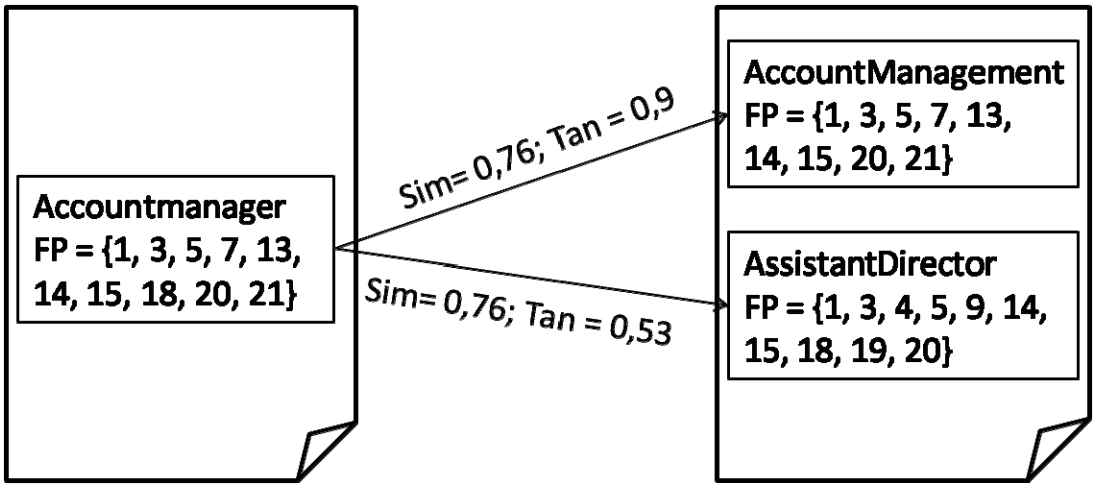


Abbildung 25: Beispiel Auflösung gleiche Ähnlichkeit

Fingerprint auf Basis spezieller Merkmale für Klassendiagramme

Die Anzahl der gefundenen Zuordnungen bei der Durchführung eines Screenings auf Basis eines speziell der Domäne angepassten Fingerprints ist leicht besser als die bei einer regulären Differenzanalyse gefundenen Korrespondenzen (vgl. Tabelle 6). Die Laufzeit dieses Ansatzes stellt dabei eine spürbare Verbesserung gegenüber der Laufzeit von SiDiff dar.

Insgesamt ist dieses Ergebnis nicht verwunderlich, da dieser Ansatz – wie beschrieben – auf Basis sehr spezieller Eigenschaften arbeitet und ein nur wenig restriktives Screening der Kandidaten verfolgt. Es ist allerdings anzumerken, dass natürlich auch bei diesem Ansatz einige der schon angesprochenen Fälle eintreten können, bei denen es zu unterschiedlichen oder nicht gefundenen Zuordnungen kommt.

Auswirkung von Filtereffekten

Bei allen vier vorgestellten Ansätzen für die Durchführung eines Screening konnte festgestellt werden, dass es unter Umständen zu Abweichungen bei den vorgenommenen Zuordnungen im Vergleich zu den von SiDiff ermittelten Korrespondenzen kommen kann. Die einzelnen Ursachen hierfür sind bei der Interpretation der verschiedenen Ansätze bereits eingehend erläutert worden.

Es ist deshalb natürlich noch insbesondere von Interesse zu untersuchen, inwieweit die auftretenden Filtereffekte sich auf die gefundenen Zuordnungen auswirken und ob dabei ansatzspezifische Unterschiede zu erkennen sind. Dazu werden die Zuordnungen der vorgestellten Ansätze eines fingerprintbasierten Screenings mit den von SiDiff durchgeführten Zuordnungen verglichen. Es werden dabei die Zuordnungen ausgehend von den Elementen des zuerst bei der Differenzierung angegebenen Dokuments (Dokument A) betrachtet. Vergleicht man auf dieser Basis die Zuordnungen von SiDiff mit den Zuordnungen eines fingerprintbasieren Ansatzes, so kann demnach einer der folgenden drei Fälle eintreten:

- Einem Element aus Dokument A ist sowohl von SiDiff als auch bei Einsatz eines Screenings jeweils das identische Element aus Dokument B zugeordnet worden (*Gleich*).
- Es findet eine Zuordnung für ein Element aus Dokument A in beiden Ansätzen statt, allerdings sind die zugeordneten Elemente nicht identisch (*Verschieden*).
- Bei Durchführung eines Screenings fand eine Zuordnung für ein Element aus Dokument A statt, während von SiDiff keine Zuordnung vorgenommen wurde (*Neu.*)

Die Anzahl der bei einem fingerprintbasierten Screening im Vergleich zu SiDiff nicht gefundenen Zuordnungen (*Nicht gefunden*) ergibt sich aus der Anzahl der Zuordnungen von SiDiff abzüglich der Anzahl der Elemente, für die in beiden Verfahren ein Partner ermittelt werden konnte. Das Ergebnis der Untersuchung ist in

Tabelle 7 zu sehen. Die Prozentzahlen geben dabei den prozentualen Anteil (gerundet auf eine Nachkommastelle) im Verhältnis zu den 755827 von SiDiff erzielten Korrespondenzen an.

	Struktur		Metrisch		Lexikalisch		Spezial	
Korrespondenzen	706196		619526		769429		765996	
Gleich	586891	77.7%	534850	70.8%	630346	83.4%	640318	84.7%
Verschieden	88751	11.7%	61782	8.2%	98890	13.1%	92721	12.3%
Neu	30554		22894		40193		32828	
Nicht gefunden	80185	10.6%	159195	21.1%	26591	3.5%	22788	3.0%

Tabelle 7: Unterschiede der Ergebnisse zu SiDiff – Fujaba (Fingerprint)

Wie dabei zu sehen ist, kommt es bei allen fingerprintbasierten Ansätzen im Vergleich zu SiDiff zu erkennbaren Filtereffekten. Das bei einem fingerprintbasierten Screening auf Basis von Softwaremetriken weniger übereinstimmende Korrespondenzen entdeckt werden, lässt sich direkt auf die aufgezeigte Grundproblematik dieses Ansatzes zurückführen. Gleiches trifft, in weniger starken Ausmaß, auch für den Ansatz auf Basis von strukturellen und allgemeinen Merkmalen zu.

Es ist festzustellen, dass bei allen vorgestellten Ansätzen (siehe Tabelle 7) eines fingerprintbasierten Screenings eine interessante Korrelation der Ergebnisse gegeben ist. Obwohl bei den einzelnen Ansätzen die Reduktion der Kandidatenmengen anhand sehr unterschiedlicher Kriterien durchgeführt wird und dementsprechend unterschiedliche Ergebnisse in den ermittelten Kandidatenmengen und Laufzeiten erzielt werden, so weist das Ausmaß der aufgetretenen Filtereffekte doch ein hohes Maß an Konstanz auf.

Das Auftreten von Filtereffekten ist auch nicht als verfahrensexklusive Eigenschaft eines fingerprintbasierten Screenings anzusehen. In seiner Diplomarbeit *Einsatz multidimensionaler Suchstrukturen zur Optimierung der Bestimmung von Dokumentdifferenzen* beschreibt *Treude* (siehe [Tre07]) einen Ansatz für die Bildung einer Kandidatenmenge anhand einer mehrdimensionalen Suchstruktur, auch S^3V-Baum genannt. In dieser Struktur werden ähnliche Elemente in Nähe zueinander angeordnet. Bei der Differenzbildung werden aussichtsreiche Kandidaten für eine Korrespondenz über ihre Nachbarschaftsbeziehung ermittelt und im Weiteren genauer auf ihre Ähnlichkeit untersucht. Obwohl dieser Ansatzes unabhängig entwickelt wurde und sich die Bildung der Kandidatenmengen unterschiedlich zu dem hier vorgestellten Verfahren eines fingerprintbasierten

Screenings gestaltet, so führt ein Vergleich der durchgeführten Zuordnungen mit den Ergebnissen von SiDiff zu fast identischen Werten (siehe Tabelle 8) betreffend der aufgetretenen Filtereffekte.

	S3V	
Korrespondenzen	743885	
Gleich	613343	81.2%
Verschieden	92466	12.2%
Neu	38076	
Nicht gefunden	50018	6.6%

Tabelle 8: Unterschiede der Ergebnisse zu SiDiff – Fujaba (S³V)

Untersucht man das Auftreten von Filtereffekten genauer, so ist weiter zu erkennen, dass das Ausmaß ihres Auftretens ein Resultat der zugrundeliegenden Testdokumente ist. Dies zeigt die Evaluation eines lexikalisch-basierten Screenings auf Grundlage weiterer Testdaten. Innerhalb der Praktischen Informatik sind im Rahmen der Diplomarbeit *Testportal zur Unterstützung rechnerbasierter Experimente (CAEW)* von *Gad* [Gad08] eine Reihe von Klassendiagrammen entstanden. Vergleicht man die Ergebnisse einer Differenzanalyse dieser Dokumente auf Basis eines lexikalisch-basierten Screenings mit den Ergebnissen von SiDiff, so ist zu sehen, dass die erzielten Ergebnisse beinahe identisch (siehe Tabelle 9) sind und Filtereffekte kaum auftreten.

	SiDiff		Lexikalisch	
Korrespondenzen	18690		18648	
Gleich			18606	99.6%
Verschieden			21	0.1%
Neu			21	
Nicht gefunden			63	0.3%

Tabelle 9: Unterschiede der Ergebnisse zu SiDiff – CAEW (Fingerprint)

Die Ergebnisse lassen den Schluss zu, dass bei dem Auftreten von Filtereffekten die Homogenität der zugrundeliegenden Dokumente eine Rolle spielt. Hält man sich den typischen Lebenszyklus eines Softwaresystems vor Augen, so ist plausibel

anzunehmen, dass Klassendiagramme, die in zeitlicher Nähe zueinander entstanden sind, einen höheren Grad an Ähnlichkeit aufweisen[30] als Klassendiagramme, die an entfernten Punkten des Lebenszyklus erzeugt wurden. Die Dokumente, die im Rahmen der Ausarbeitung von *Gad* entstanden, spiegeln einen entsprechend kürzeren Entwicklungszeitraum wider, als die Testdokumente aus dem Projekt *Fujaba*.

Letztendlich beruht jedweder Ansatz einer Kandidatenbildung auf einer Fokussierung auf Kriterien als Grundlage für die Bildung der Menge an aussichtsreichen Kandidaten. Eine solche Fokussierung führt zu Unterschieden in den Kandidatenmengen und in letzter Konsequenz zu Unterschieden in den gefundenen Korrespondenzen. Wie bei Klassendiagrammen zu sehen, wird dieser Effekt durch eine Berechnung der Ähnlichkeit nach komplexen Kriterien, sowie dem Vorkommen von *Teil-von*-Beziehungen verstärkt. Wie stark diese Filtereffekte allerdings zu Tage treten, ist entscheidend von Eigenschaften der zu differenzierenden Dokumente abhängig. Da dabei insbesondere die Homogenität eine Rolle spielt, ist für eine Abschätzung die Formulierung eines Ansatzes nötig, der die erwartete Differenz der zugrundeliegenden Dokumente ermittelt. Auf dieser Basis ließe sich dann das zu erwartende Ausmaß des Effekts abschätzen. Da eine solche Analyse als nicht trivial anzusehen ist und den Rahmen dieser Ausarbeitung sprengen würde, ist auf die Ermittlung verzichtet worden.

Fazit

Wie gezeigt wurde eignen sich allgemeine und strukturelle Merkmale von UML-Klassendiagrammen nur bedingt um eine Verbesserung der Laufzeit der Differenzanalyse zu erzielen. Zum einen gibt es - wie bei UML-Aktivitätsdiagrammen - eine Menge an Elementtypen, die sich hinsichtlich dieser Art von Eigenschaften nicht unterscheidet. Hingegen ist für Elementtypen, die Unterschiede in dieser Beziehung aufweisen, festzustellen, dass die Änderungen mitunter in einem derartigen Ausmaß auftreten, dass eine sinnvolle Unterscheidung nicht immer möglich ist. Ebenso erwies sich ein Einsatz modellspezifischer Kenngrößen, sogenannter Softwaremetriken, als ein nicht geeigneter Ansatz für die Durchführung eines fingerprintbasierten Screenings.

Es konnte auch in dieser Domäne gezeigt werden, dass sich insbesondere namensspezifische Eigenschaften für die Bildung von Kandidatenmengen nutzen lassen. Es ist demnach anzunehmen, dass alle Modelltypen gut für den Einsatz eines

[30] *Dabei ist allerdings auch zu bedenken, dass Versionen die zwar zeitlich nahe zusammen liegen, aber einen Sprung in der Entwicklung darstellen (typischer an einem Sprung in der Versionsnummer zu erkennen), trotzdem größere inhaltliche Veränderungen aufweisen können.*

fingerprintbasierten Screenings geeignet sind, die auf eine distinkte, namensbasierte Identifizierung ihrer Elemente zurückgreifen. Es konnte weiterhin festgestellt werden, dass auch ein Ansatz auf Basis eines äußert differenzierten Fingerprints zu guten Ergebnissen führt, allerdings keine nennenswerte Verbesserung gegenüber einem lexikalisch-basierten Fingerprint erzielt.

Bei einer Analyse der ermittelten Korrespondenzen wurde weiterhin festgestellt, dass durch Bildung einer Kandidatenmenge mitunter Situationen auftreten, die zu Unterschieden in den gefundenen Zuordnungen führen. Bei genauerer Untersuchung dieses Phänomens zeigte sich, dass das Ausmaß der auftretenden Filtereffekte weitestgehend unabhängig von den gewählten Kriterien für die Durchführung eines Screenings ist. Vielmehr ließ sich der beobachtete Effekt auf Eigenschaften der zugrundeliegenden Testdaten zurückzuführen. Liegen homogene Dokumente vor, so erzielt ein Screening fast identische Ergebnisse im Vergleich zu SiDiff. Wiesen die Dokumente allerdings deutliche Unterschiede auf, beispielsweise weil sie einen längeren Entwicklungszeitraum widerspiegeln, kommt es zu bemerkbaren Filtereffekten. Eine allgemeine Abschätzung für das Ausmaß von Filtereffekten kann ohne eine weitere Analyse der Dokumente nicht aufgestellt werden.

Zum Abschluss des Kapitels sei noch auf Tabelle 10 hingewiesen. Dort sind die Ergebnisse einer Dokumentdifferenzierung von SiDiff unter Berücksichtigung der von *Wehren* vorgeschlagenen Hashingphase zu sehen. Weiterhin ist zusätzlich ein fingerprintbasiertes Screening nach dem vorgestellten lexikalischen Ansatz durchgeführt worden. Wie an den Ergebnissen zu sehen ist, führt das Screening einer Kandidatenmenge auf Basis eines lexikalisch-basierten Fingerprints auch in diesem Fall zu einer deutlichen Verbesserung der Laufzeit. Selbstverständlich treten aber auch hier die beschriebenen Unterschiede in den vorgenommenen Zuordnungen auf.

	SiDiff Hash	***Lexikalisch Hash***
Zuordnungen	781996	-4.0%
Zeit	206min	-78.2%

Tabelle 10: Übersicht Ergebnisse Klassendiagramme (Hashing)

5.3 MATLAB/Simulink-Diagramme

MATLAB/Simulink ist eine Software aus dem Hause *MathWorks*[31]. Sie dient der Modellierung, Simulation sowie Analyse dynamischer Systeme (siehe [Ang05], S. 267). Ein zu simulierendes System wird als ein Signalflussgraph dargestellt (vgl. [Pie05], S. 99), wobei die Funktionalität durch verknüpfte Funktionsblöcke abgebildet ist. Es ist weiterhin möglich, ein Modell in logisch zusammenhängende Systeme zu gliedern, beispielsweise um funktionale Komponenten zu bilden.

MATLAB/Simulink bietet im Gegensatz zu den bisher besprochenen Modellen nicht nur einen umfangreicheren Basisbestand an Elementen, sondern es ist darüberhinaus möglich, eigene Elemente zu definieren. Somit tritt bei dieser Domäne insbesondere eine neue Problematik auf und es ist zu überlegen, inwiefern das Vorhandensein unbekannter Elementtypen innerhalb eines fingerprintbasierten Screenings berücksichtig werden kann.

Evaluationsansatz für MATLAB/Simulink-Diagramme

Aufgrund der modellspezifischen Möglichkeit, eigene Blöcke zu definieren, ist es nicht möglich, Annahmen über elementimmanente Charakteristika zu treffen. Die Möglichkeit, Elemente anhand von Charakteristika zu unterscheiden, ist aber bereits als Grundvoraussetzung für die Bildung einer Kandidatenmenge aufgezeigt worden.

Die dominierenden Elementtypen des Modells sind die Verbindungelemente, welche die Wege der Datenströme zwischen den einzelnen Blöcken des modellierten Systems festlegen. Allerdings findet sich auch an dieser Stelle kein Ansatzpunkt für ein effektives Screening. Ob zwischen zwei Verbindungselementen eine Korrespondenz vorliegt beruht maßgeblich darauf, ob zwischen den angrenzenden Elementen bereits eine Korrespondenz vorliegt (siehe *Anhang B III: Simulink.compare.xml (Auszug)*). Die Ermittlung von Korrespondenzen geschieht allerdings erst bei Durchführung der Differenzanalyse. Da ein fingerprintbasiertes Screening vor der Berechnung der Ähnlichkeitswerte ansetzt, scheitert das Verfahren an dieser Stelle.

Selbst der vorgestellte strukturelle Ansatz zur Bildung von Fingerprints erweist sich für diesen Modelltyp als problematisch. Für den vorliegenden Diagrammtyp wird ein *Top-Down*-Ansatz bei der Durchführung einer Differenzanalyse genutzt. Für die zu analysierenden Diagramme werden zuerst die Korrespondenzen zwischen

[31] *http://www.mathworks.com/*

den vorkommenden Systemen ermittelt. Weitere Elemente werden nur auf ihre Ähnlichkeit untersucht, wenn sie korrespondierende Elternelemente besitzen. Da sich die abgebildeten Eigenschaften eines Systems in einem Fingerprint allerdings von Dokument zu Dokument stark verändern, werden diese in der Regel durch ein Screening gefiltert und es werden keine weiteren Elemente mehr auf Ähnlichkeit untersucht.

Eine derartige Bedingung lässt sich – ebenso wie die bereits angesprochene Problematik für Verbindungen - nicht mit dem vorgestellten Fingerprintverfahren nachbilden.

Aus den genannten Gründen konnte kein sinnvoller Evaluationsansatz für Diagramme aus dieser Domäne formuliert werden und eine weitere Betrachtung entfällt im Rahmen dieser Arbeit.

Fazit

Anhand Eigenschaften der untersuchten Domäne MATLAB/Simulink konnte gezeigt werden, unter welchen Bedingungen der Einsatz eines fingerprintbasierten Screenings an seine Grenzen stößt. Greift die Differenzanalyse bei Berechnung der Ähnlichkeit zwischen Elementen auf bereits ermittelte Korrespondenzen zurück, so lässt sich dieser Umstand nicht in einem Fingerprint abbilden. Ein fingerprintbasiertes Screening ist einzig ein statischer Blick auf die in dem Modell abgebildeten Elemente und es lassen sich keine dynamischen Kriterien berücksichtigen. Für das vorgestellte Verfahren ist auch problematisch, dass es nicht möglich ist, charakteristische Merkmale für alle Elemente zu definieren. Das Auftreten unbekannter Elementtypen zu berücksichtigen, ist nicht umsetzbar.

6. Zusammenfassung und Ausblick

Zusammenfassung

In dieser Arbeit ist ein Ansatz aus der Bioinformatik für ein Screening molekularer Datenbanken (siehe *Kapitel 3. Fingerprintbasiertes Screening*) vorgestellt worden. Dabei wird auf Basis charakteristischer Eigenschaften die Menge an chemischen Strukturen aus einer Datenbank ermittelt, die als aussichtsreiche Kandidaten für eine Suchanfrage anzusehen sind. Es galt im Weiteren zu untersuchen, ob sich ein derartiger Ansatz bei einem Einsatz in der Differenzanalyse von Dokumenten ebenfalls bewährt. Ziel des Transfers war es, die Laufzeit einer Differenzanalyse zu verringern. Insbesondere galt festzustellen, welche Eigenschaften von Modelltypen und Elementen sich für den Einsatz eines derartigen Screenings eignen.

Wie bei einer genauen Untersuchung der beiden Anwendungsgebiete (vgl. *Kapitel 4.2 Unterschiede der Einsatzgebiete)* festgestellt wurde, treten doch einige deutliche Unterschiede zu Tage. Zum einen arbeitet die Differenzanalyse von Dokumenten auf Ebene der Elemente, während ein Screening nach chemischen Strukturen auf Basis des Äquivalents zu Dokumenten ansetzt. Es konnte weiterhin gezeigt werden, dass sich eine Abbildung von zusammenhängenden Strukturen, respektive Fragmenten (siehe *Kapitel 3.3 Hashed Fingerprints*), im Rahmen des neuen Einsatzkontexts nicht sinnvoll für eine Unterscheidung einzelner Elemente nutzen lässt. Auch die Abbildung von den in der Differenzanalyse häufig genutzten Eigenschaften zur Bestimmung eines Ähnlichkeitswerts auf eine Bitfolge erwies sich als problematisch. Der kritischste Unterschied der beiden Einsatzgebiete ist allerdings die Genauigkeit, mit der eine Kandidatenmenge ermittelt werden kann. Im originären Kontext ist davon auszugehen, dass potentielle Kandidaten exakt die Eigenschaften des gesuchten Musters erfüllen. Bei einer Differenzierung von Dokumenten wird hingegen implizit davon ausgegangen, dass mitunter sehr wohl Unterschiede zwischen den Eigenschaften korrespondierender Elemente auftreten. Auf Grund dessen ist der Einsatz eines Fingerprintverfahrens einzig als heuristischer Ansatz zu verstehen, der keine Garantie auf eine genaue Bestimmung der Kandidatenmengen geben kann. Es konnte ebenso gezeigt werden, dass ein

Screening auf Basis von Fingerprints allerdings grundsätzlich für die Lösung der beschriebenen Aufgabenstellung geeignet ist.

Als positive modellspezifische Eigenschaften für ein Screening sind eine entsprechende Größe der Dokumente, das Vorhandensein eines oder mehrerer dominierender Elementtypen, sowie eine Berechnung der Ähnlichkeit zwischen Elementen anhand von komplexen Ähnlichkeitskriterien identifiziert worden.

Bei der Untersuchung der verschiedenen für die Evaluierung des Verfahrens genutzten Modelltypen stellte sich heraus, dass insbesondere Domänen für den Einsatz eines fingerprintbasierten Screenings geeignet sind, die als spezifische Eigenschaft auf eine Identifizierung der Elemente anhand von Namen zurückgreifen (vgl. *Kapitel 5.2.2 Aktivitätsdiagramme, lexikalisch* und *Kapitel 5.2.3 Klassendiagramme, lexikalisch*).

Strukturelle und allgemeine Eigenschaften hingegen, wie sie beispielsweise aus der Repräsentation der Elemente innerhalb des vorgestellten internen Datenmodells für die Differenzanalyse entstehen, eignen sich nicht oder nur bedingt als Ansatz für eine Reduktion der Kandidatenmengen. Zum einen sind die durch das Modell festgelegten Strukturen oftmals so starr, dass sich auf ihrer Basis keine Differenzierung zwischen den einzelnen Elementen eines Typs (vgl. *Kapitel 5.2.2 Aktivitätsdiagramme, Struktur*) erzielen lässt. Bei anderen Elementtypen kommt es hingegen oft zu so starken Änderungen, dass sich ein Erkennen ähnlicher Elemente auf Grund dessen schwierig gestaltet (vgl. *Kapitel 5.2.3 Klassendiagramme, Struktur*).

Bei der Analyse von UML-Klassendiagrammen erwies sich eine Nutzung modellspezifischer Metriken als nicht geeignet für die Bildung einer Menge an aussichtsreichen Kandidaten (siehe *Kapitel 5.2.3 Klassendiagramme, Metrisch*). Während Softwaremetriken zwar durchaus als Qualitätsmaß für Softwaresysteme geeignet sind und Rückschlüsse auf Eigenschaften wie Wartbarkeit, Erweiterbarkeit und Verständlichkeit des Systems ermöglichen, ändern sich die entsprechenden Kennzahlen zwischen verschiedenen Versionen eines Dokuments oftmals zu stark um ähnliche Elemente zu identifizieren. Das Problem wird verstärkt, da nicht alle Softwaremetriken in UML-Klassendiagrammen enthalten sind und zwischen den einzelnen Metriken Abhängigkeiten existieren.

Bei der Evaluierung des vorgestellten Verfahrens eines fingerprintbasierten Screenings konnte das Problem der unscharfen Bildung von Kandidatenmengen genauer aufgezeigt werden. Insbesondere bei nicht homogenen Testdaten kommt es zu dem beschriebenen Phänomen der Filtereffekte (siehe *Kapitel 5.2.3 Klassendiagramme, Auswirkung von Filtereffekten*). Hintergrund ist, dass die Bildung einer Kandidatenmenge immer auf einer Selektion von Elementen nach ausgewählten Kriterien basiert. Da nicht alle Kriterien berücksichtigt bzw. exakt

abgebildet werden können, kann es somit zu einer Filterung von Elementen kommen, die ansonsten als Korrespondenz erkannt werden. Es konnte weiterhin gezeigt werden, dass dies keine verfahrensexklusive Eigenschaft darstellt, sondern auch andere Ansätze der Kandidatenbildung von dieser Problematik betroffen sind. Bei einer genaueren Untersuchung der bei der Evaluation von UML-Klassendiagrammen aufgetretenen Filtereffekte wurde festgestellt, dass sie - unabhängig von dem gewählten Ansatz des Screenings - ein hohes Maß an Konstanz aufweisen und maßgeblich von den genutzten Dokumenten abhängig sind.

Bei der Evaluierung des Fingerprintverfahrens in der Domäne MATLAB/Simulink (siehe *Kapitel 5.3* MATLAB/Simulink-Diagramme*)* konnte gezeigt werden, unter welchen Bedingungen das vorgestellte Verfahren an seine Grenzen stößt. Wenn in einer Domäne unbekannte Elementtypen auftreten gibt es keinen Ansatzpunkt, um charakteristische Eigenschaften in einem Fingerprint abzubilden. Auch können in einem fingerprintbasierten Screening keine Kriterien berücksichtigt werden, die sich erst zur Laufzeit ergeben.

Abschließendes Fazit

Die Evaluation des Verfahrens hat gezeigt, dass sich mittels eines geeignet durchgeführten Screenings die Laufzeit einer Differenzanalyse mitunter drastisch verringern lässt. Auch wenn gezeigt werden konnte, dass sich nicht in allen Domänen ein Screening erfolgreich einsetzen lässt. Allerdings ist abschließend der heuristische Ansatz des Verfahrens erneut kritisch hervorzuheben. Denkt man an die bei der Evaluierung zu Tage getretenen Filtereffekte, so stellt sich die Frage, ob ein Ansatz der auf eine Bildung Kandidatenmengen abzielt, überhaupt in dem vorliegenden Anwendungsgebiet als sinnvoll einzuschätzen ist. Schließlich ist die Zielsetzung einer Dokumentendifferenzierung die Differenz zwischen den Dokumenten so exakt wie möglich zu bestimmen. Die Bildung von Kandidatenmengen ist aber, wie gezeigt werden konnte, immer mit einer gewissen Unschärfe verbunden.

Es ist auf Grund dessen fraglich, ob der Einsatz eines fingerprintbasierten Screenings bei der Differenzanalyse von technischen Dokumenten weiter verfolgt werden sollte.

Ausblick

Derzeit wird untersucht, inwiefern sich ein Einsatz von Fingerprints in anderen Bereichen der Dokumentdifferenzierung nutzen lässt. Insbesondere ist ein Einsatz eines angepassten Verfahrens im Rahmen der *Clon Detection* ins Auge gefasst. Dabei wird versucht, in den Fingerprint eines Elements auch Eigenschaften von Nachbarelementen geeignet zu codieren. Ziel ist es im Weiteren durch Einsatz einfacher Verknüpfungsoperationen anhand der Fingerprints effizient identische Teilabschnitte in Modellen zu finden. Da potentielle Klone identische Fingerprints aufweisen, scheint der Einsatz von Fingerprints in diesem Anwendungskontext mit weniger Problemen verbunden zu sein, als in dem in dieser Arbeit vorgestellten Einsatzgebiet.

Anhang

Anhang A: Fingerprints

A I: DTD Blueprints

```
<?xml version="1.0" encoding="ISO-8859-1"?>

      <!ELEMENT CharactersOfAttributeValue EMPTY>
      <!ATTLIST CharactersOfAttributeValue
            nodetype     CDATA      #REQUIRED
            mapping      CDATA      #REQUIRED
            attribute    CDATA      #REQUIRED>
      <!ELEMENT CharactersOfAttributeValueNeighbor EMPTY>
      <!ATTLIST CharactersOfAttributeValueNeighbor
            nodetype     CDATA      #REQUIRED
            mapping      CDATA      #REQUIRED
            attribute    CDATA      #REQUIRED
            edgetype    (reference|
                         nesting)   #REQUIRED
            edgename     CDATA      #REQUIRED
            direction   (incoming|
                         outgoing|
                         any)             #REQUIRED>
      <!ELEMENT LengthOfAttributeValue EMPTY>
      <!ATTLIST LengthOfAttributeValue
            nodetype     CDATA      #REQUIRED
            mapping      CDATA      #REQUIRED
            range        CDATA      #REQUIRED
            attribute    CDATA      #REQUIRED>
      <!ELEMENT LengthOfAttributeValueNeighbor EMPTY>
      <!ATTLIST LengthOfAttributeValueNeighbor
            nodetype     CDATA      #REQUIRED
            mapping      CDATA      #REQUIRED
            range        CDATA      #REQUIRED
            attribute    CDATA      #REQUIRED
            edgetype     (reference|
                        nesting)    #REQUIRED
            edgename     CDATA      #REQUIRED
            direction    (incoming|
                          outgoing|
                          any)      #REQUIRED>
```

```
<!ELEMENT LengthOfNeighborNodeTypeNames EMPTY>
<!ATTLIST LengthOfNeighborNodeTypeNames
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED
      direction    (incoming|
                    outgoing|
                    any)      #REQUIRED>
<!ELEMENT NumberOfAttributes EMPTY>
<!ATTLIST NumberOfAttributes
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED>
<!ELEMENT NumberOfAttributeValue EMPTY>
<!ATTLIST NumberOfAttributeValue
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED
      attribute    CDATA      #REQUIRED>
<!ELEMENT NumberOfAttributeValuesNeighborByEdge EMPTY>
<!ATTLIST NumberOfAttributeValuesNeighborByEdge
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED
      value        CDATA      #REQUIRED
      attribute    CDATA      #REQUIRED
      edgetype     (reference|
                    nesting)  #REQUIRED
      edgename     CDATA      #REQUIRED
      direction    (incoming|
                    outgoing|
                    any)      #REQUIRED>
<!ELEMENT NumberOfNeighborNodes EMPTY>
<!ATTLIST NumberOfNeighborNodes
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED
      direction    (incoming|
                    outgoing|
                    any)      #REQUIRED>
<!ELEMENT NumberOfNeighborNodeTypes EMPTY>
<!ATTLIST NumberOfNeighborNodeTypes
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED
      direction    (incoming|
                    outgoing|
                    any)      #REQUIRED>
<!ELEMENT TestForAttributeValue EMPTY>
<!ATTLIST TestForAttributeValue
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      value        CDATA      #REQUIRED
      attribute    CDATA      #REQUIRED
```

```
      match       (true|
                    false)    #REQUIRED>
<!ELEMENT TestForAttributeValueNeighborByEdge EMPTY>
<!ATTLIST TestForAttributeValueNeighborByEdge
      nodetype     CDATA      #REQUIRED
      mapping      CDATA      #REQUIRED
      value        CDATA      #REQUIRED
      attribute    CDATA      #REQUIRED
      match       (true|
                    false)    #REQUIRED
      edgetype    (reference|
                    nesting)  #REQUIRED
      edgename     CDATA      #REQUIRED
      direction    (incoming|
                    outgoing|
                    any)      #REQUIRED>
<!ELEMENT ActivityEdgeNeighbor EMPTY>
<!ATTLIST ActivityEdgeNeighbor
      mapping      CDATA      #REQUIRED>
<!ELEMENT IsGetter EMPTY>
<!ATTLIST IsGetter
      mapping      CDATA      #REQUIRED>
<!ELEMENT IsSetter EMPTY>
<!ATTLIST IsSetter
      mapping      CDATA      #REQUIRED>
<!ELEMENT NumberOfAccessors EMPTY>
<!ATTLIST NumberOfAccessors
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED>
<!ELEMENT NumberOfAttributesForClass EMPTY>
<!ATTLIST NumberOfAttributesForClass
      mapping      CDATA      #REQUIRED
      visibility   (public|
                    private|
                    protected|
                    any)      #REQUIRED
      range        CDATA      #REQUIRED>
<!ELEMENT NumberOfGetters EMPTY>
<!ATTLIST NumberOfGetters
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED>
<!ELEMENT NumberOfSetters EMPTY>
<!ATTLIST NumberOfSetters
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED>
<!ELEMENT NumberOfMethods EMPTY>
<!ATTLIST NumberOfMethods
      mapping      CDATA      #REQUIRED
      range        CDATA      #REQUIRED
      visibility   (public|
                    private|
                    protected|
                    any)      #REQUIRED>
<!ELEMENT TestForAccessor EMPTY>
```

```
      <!ATTLIST TestForAccessor
            mapping      CDATA      #REQUIRED
            range        CDATA      #REQUIRED>
      <!ELEMENT TestForAccessorName EMPTY>
      <!ATTLIST TestForAccessorName
            mapping      CDATA      #REQUIRED>

      <!ELEMENT TestForVisibility EMPTY>
      <!ATTLIST TestForVisibility
            nodetype     CDATA      #REQUIRED
            mapping      CDATA      #REQUIRED>

      <!ELEMENT blueprint (
                  CharactersOfAttributeValue* ,
                  CharactersOfAttributeValueNeighbor* ,
                  LengthOfAttributeValue* ,
                  LengthOfAttributeValueNeighbor* ,
                  LengthOfNeighborNodeTypeNames* ,
                  NumberOfAttributes* ,
                  NumberOfAttributeValue* ,
                  NumberOfAttributeValuesNeighborByEdge* ,
                  NumberOfNeighborNodes* ,
                  NumberOfNeighborNodeTypes* ,
                  TestForAttributeValue* ,
                  TestForAttributeValueNeighborByEdge* ,
                  ActivityEdgeNeighbor* ,
                  IsGetter* ,
                  IsSetter* ,
                  NumberOfAccessors* ,
                  NumberOfAttributesForClass* ,
                  NumberOfGetters* ,
                  NumberOfSetters* ,
                  NumberOfMethods* ,
                  TestForAccessor*,
                  TestForAccessorName* ,
                  TestForVisibility*)>
```

A II: Fingerprint über strukturelle und allgemeine Eigenschaften

```
<blueprint>
      <!—Any-->
      <NumberOfAttributes
            nodetype="any"
            range="50"
            mapping="0"/>
      <NumberOfNeighborNodes
            nodetype="any"
            direction="incoming"
            range="50"
```

```
            mapping="0"/>
      <NumberOfNeighborNodes
            nodetype="any"
            direction="outgoing"
            range="50"
            mapping="0"/>
      <NumberOfNeighborNodeTypes
            nodetype="any"
            direction="incoming"
            range="50"
            mapping="50"/>
      <NumberOfNeighborNodeTypes
            nodetype="any"
            direction="outgoing"
            range="50"
            mapping="50"/>
      <LengthOfNeighborNodeTypeNames
            nodetype="any"
            direction="incoming"
            range="50"
            mapping="50"/>
      <LengthOfNeighborNodeTypeNames
            nodetype="any"
            direction="outgoing"
            range="50"
            mapping="50"/>
</blueprint>
```

A III: Fingerprint lexikalische Merkmale (UML-Aktivitätsdiagramme)

```
<blueprint>
      <!--CallAction->
      <CharactersOfAttributeValue
            nodetype="callAction"
            mapping="0"
            attribute="name"/>
      <!--controlNode-->
      <CharactersOfAttributeValue
            nodetype="controlNode"
            mapping="0"
            attribute="name"/>
      <!--activityEdge-->
      <CharactersOfAttributeValueNeighbor
            nodetype="activityEdge"
            mapping="0"
            edgename="target"
            edgetype="reference"
            direction="any"
            attribute="name"/>
```

```
      <CharactersOfAttributeValueNeighbor
            nodetype="activityEdge"
            mapping="50"
            edgename="source"
            edgetype="reference"
            direction="any"
            attribute="name"/>
</blueprint>
```

A IV: Fingerprint Softwaremetriken (UML-Klassendiagramme)

```
<blueprint>
      <!--Class-->
      <NumberOfGetters
            mapping="50"
            range="50"/>
      <NumberOfSetters
            mapping="50"
            range="50"/>
      <NumberOfMethods
            mapping="50"
            range="50"
            visibility="any"/>
      <NumberOfAttributesForClass
            mapping="50"
            range="50"
            visibility="any"/>
      <NumberOfAttributeValuesNeighborByEdge
            nodetype="class"
            direction="any"
            edgename="methods"
            edgetype="nesting"
            attribute="visibility"
            value="public"
            range="100"
            mapping="50"/>
      <NumberOfAttributeValuesNeighborByEdge
            nodetype="class"
            direction="any"
            edgename="methods"
            edgetype="nesting"
            attribute="visibility"
            value="protected"
            range="100"
            mapping="50"/>
      <NumberOfAttributeValuesNeighborByEdge
            nodetype="class"
            direction="any"
            edgename="methods"
```

```
            edgetype="nesting"
            attribute="visibility"
            value="private"
            range="100"
            mapping="50"/>
      <NumberOfAttributeValuesNeighborByEdge
            nodetype="class"
            direction="any"
            edgename="attrs"
            edgetype="nesting"
            attribute="visibility"
            value="public"
            range="150"
            mapping="50"/>
      <NumberOfAttributeValuesNeighborByEdge
            nodetype="class"
            direction="any"
            edgename="attrs"
            edgetype="nesting"
            attribute="visibility"
            value="protected"
            range="150"
            mapping="50"/>
      < NumberOfAttributeValuesNeighborByEdge
            nodetype="class"
            direction="any"
            edgename="attrs"
            edgetype="nesting"
            attribute="visibility"
            value="private"
            range="150"
            mapping="50"/>
</blueprint>
```

A V: Fingerprint lexikalische Merkmale (UML-Klassendiagramme)

```
<blueprint>
      <!--Package-->
      <CharactersOfAttributeValue
            nodetype="package"
            mapping="0"
            attribute="name"/>

      <!--Class-->
      <CharactersOfAttributeValue
            nodetype="class"
            mapping="0"
            attribute="name"/>
      <LengthOfAttributeValueNeighbor
            nodetype="class"
            mapping="30"
```

```
      edgetype="nesting"
      edgename="methods"
      attribute="name"
      range="30"
      direction="any"/>
<LengthOfAttributeValueNeighbor
      nodetype="class"
      mapping="60"
      edgetype="nesting"
      edgename="attrs"
      attribute="name"
      range="30"
      direction="any"/>
<LengthOfAttributeValueNeighbor
      nodetype="class"
      mapping="100"
      edgetype="reference"
      edgename="typeOf"
      attribute="name"
      range="50"
      direction="any"/>

<!--Attribute-->
<CharactersOfAttributeValue
      nodetype="attribute"
      mapping="0"
      attribute="name"/>
<CharactersOfAttributeValueNeighbor
      nodetype="attribute"
      mapping="30"
      attribute="name"
      edgetype="nesting"
      edgename="attrs"
      direction="any"/>
<CharactersOfAttributeValueNeighbor
      nodetype="attribute"
      mapping="60"
      attribute="name"
      edgetype="reference"
      edgename="typeof"
      direction="any"/>

<!--Operation-->
<CharactersOfAttributeValue
      nodetype="operation"
      mapping="0"
      attribute="name"/>
<CharactersOfAttributeValueNeighbor
      nodetype="operation"
      mapping="30"
      attribute="name"
      edgetype="nesting"
      edgename="methods"
      direction="any"/>
```

```
        <CharactersOfAttributeValueNeighbor
              nodetype="operation"
              mapping="30"
              attribute="name"
              edgetype="nesting"
              edgename="parameters"
              direction="any"/>

        <!--Parameter-->
        <CharactersOfAttributeValue
              nodetype="parameter"
              mapping="0"
              attribute="name"/>
        <CharactersOfAttributeValueNeighbor
              nodetype="parameter"
              mapping="30"
              attribute="name"
              edgetype="nesting"
              edgename="parameters"
              direction="any"/>
        <CharactersOfAttributeValueNeighbor
              nodetype="parameter"
              mapping="60"
              attribute="name"
              edgetype="reference"
              edgename="paramType"
              direction="any"/>

        <!--Stereotype-->
        <CharactersOfAttributeValue
              nodetype="stereotype"
              mapping="0"
              attribute="name"/>

        <!--Datatype-->
        <CharactersOfAttributeValue
              nodetype="datatype"
              mapping="0"
              attribute="name"/>
        <LengthOfAttributeValueNeighbor
              nodetype="datatype"
              mapping="100"
              edgetype="reference"
              edgename="typeof"
              attribute="name"
              range="50"
              direction="any"/>

</blueprint>
```

A VI: Fingerprint spezielle Eigenschaften (UML-Klassendiagramme)

```
<blueprint>

      <!--Class-->
      <CharactersOfAttributeValue
            nodetype="class"
            mapping="0"
            attribute="name"/>
      <LengthOfAttributeValueNeighbor
            nodetype="class"
            mapping="30"
            edgetype="nesting"
            edgename="methods"
            attribute="name"
            range="30"
            direction="any"/>
      <LengthOfAttributeValueNeighbor
            nodetype="class"
            mapping="60"
            edgetype="nesting"
            edgename="attrs"
            attribute="name"
            range="30"
            direction="any"/>
      <TestForVisibility
            nodetype="class"
            mapping="100"/>
      <NumberOfGetters
            mapping="110"
            range="50"/>
      <NumberOfSetters
            mapping="110"
            range="50"/>
      <NumberOfMethods mapping="110"
            range="50"
            visibility="any"/>
      <NumberOfAttributesForClass
            mapping="110"
            range="50"
            visibility="any" />
      <NumberOfNeighborNodes
            nodetype="class"
            mapping="110"
            edgetype="nesting"
            edgename="typeof"
            attribute="name"
            range="50"
```

```
      direction="any"/>
<NumberOfAttributeValuesNeighborByEdge
      nodetype="class"
      direction="any"
      edgename="methods"
      edgetype="nesting"
      attribute="visibility"
      value="public"
      range="160"
      mapping="30"/>
<NumberOfAttributeValuesNeighborByEdge
      nodetype="class"
      direction="any"
      edgename="methods"
      edgetype="nesting"
      attribute="visibility"
      value="protected"
      range="160"
      mapping="30"/>
<NumberOfAttributeValuesNeighborByEdge
      nodetype="class"
      direction="any"
      edgename="methods"
      edgetype="nesting"
      attribute="visibility"
      value="private"
      range="160"
      mapping="30"/>
<NumberOfAttributeValuesNeighborByEdge
      nodetype="class"
      direction="any"
      edgename="attrs"
      edgetype="nesting"
      attribute="visibility"
      value="public"
      range="210"
      mapping="30"/>
<NumberOfAttributeValuesNeighborByEdge
      nodetype="class"
      direction="any"
      edgename="attrs"
      edgetype="nesting"
      attribute="visibility"
      value="protected"
      range="210"
      mapping="30"/>
<NumberOfAttributeValuesNeighborByEdge
      nodetype="class"
      direction="any"
      edgename="attrs"
      edgetype="nesting"
      attribute="visibility"
      value="private"
      range="210"
```

```
      mapping="30"/>

<!--Attribute-->
<CharactersOfAttributeValue
      nodetype="attribute"
      mapping="0"
      attribute="name"/>
<CharactersOfAttributeValueNeighbor
      nodetype="attribute"
      mapping="30"
      attribute="name"
      edgetype="nesting"
      edgename="attrs"
      direction="any"/>
<CharactersOfAttributeValueNeighbor
      nodetype="attribute"
      mapping="60"
      attribute="name"
      edgetype="reference"
      edgename="typeof"
      direction="any"/>

<!--Operation-->
<TestForAccessorName
      mapping="0" />
<CharactersOfAttributeValueNeighbor
      nodetype="operation"
      mapping="30"
      attribute="name"
      edgetype="nesting"
      edgename="methods"
      direction="any"/>
<CharactersOfAttributeValueNeighbor
      nodetype="operation"
      mapping="30"
      attribute="name"
      edgetype="nesting"
      edgename="parameters"
      direction="any"/>

<!--Parameter-->
<CharactersOfAttributeValue
      nodetype="parameter"
      mapping="0"
      attribute="name"/>
<CharactersOfAttributeValueNeighbor
      nodetype="parameter"
      mapping="30"
      attribute="name"
      edgetype="nesting"
      edgename="parameters"
      direction="any"/>
<CharactersOfAttributeValueNeighbor
      nodetype="parameter"
```

```
            mapping="60"
            attribute="name"
            edgetype="reference"
            edgename="paramType"
            direction="any"/>

      <!--Stereotype-->
      <CharactersOfAttributeValue
            nodetype="stereotype"
            mapping="0"
            attribute="name"/>

      <!--Datatype-->
      <CharactersOfAttributeValue
            nodetype="datatype"
            mapping="0"
            attribute="name"/>

      <!--Package-->
      <CharactersOfAttributeValue
            nodetype="package"
            mapping="0"
            attribute="name"/>

</blueprint>
```

Anhang B: Vergleichskonfigurationen des SiDiff-Algorithmus

B I: umlActivityCompareConfig.xml (Auszug)

```
<SiDiffCompare>
   <NodeType name="callAction">
      <Configuration threshold="0.5"/>
      <CompareFunctions>
      <CompareFunction
            class="StringAttributeUsingLcsCS"
            weight="0.6"
            parameter="name"/>
      </CompareFunctions>
   </NodeType>
   …
   <NodeType name="controlNode">
      <Configuration threshold="0.5" />
      <CompareFunctions>
      <CompareFunction
            class="StringAttributeUsingLcsCS"
            weight="0.5"
            parameter="name"/>
      <CompareFunction
            class="StringAttributeUsingEqualsCS"
            weight="0.5"
            parameter="kind"/>
      </CompareFunctions>
   </NodeType>
   …
   <NodeType name="activityEdge">
      <Configuration threshold="0.6" />
      <CompareFunctions>
      <CompareFunction
            class="StringAttributeUsingLcsCS"
            weight="0.1"
            parameter="name"/>
      <CompareFunction
            class="StringAttributeUsingEqualsCS"
            weight="0.1"
            parameter="guard"/>
      <CompareFunction
            class="OutgoingReferenceMatchedOrSimilar"
            weight="0.4"
            parameter="target"/>
      <CompareFunction
            class="OutgoingReferenceMatchedOrSimilar"
            weight="0.4"
            parameter="source"/>
```

```
      </CompareFunctions>
   </NodeType>
</SiDiffCompare>
```

B II: umlClassCompareConfig.xml (Auszug)

```
<SiDiffCompare>
   <NodeType name="attribute">
      <Configuration threshold="0.6"/>
      <CompareFunctions>
      <CompareFunction
               class="StringAttributeUsingLcsCS"
               weight="0.5"
               parameter="name"/>
      <CompareFunction
               class="StringAttributeUsingEqualsCS"
               weight="0.04"
               parameter="visibility"/>
      <CompareFunction
               class="StringAttributeUsingEqualsCS"
               weight="0.04"
               parameter="changeability"/>
      <CompareFunction
               class="StringAttributeUsingEqualsCS"
               weight="0.04"
               parameter="ownerScope"/>
      <CompareFunction
               class="OutgoingReferenceMatchedOrSimilar"
               weight="0.2"
               parameter="typeof"/>
      <CompareFunction
               class="ParentsMatched"
               weight="0.15"/>
      </CompareFunctions>
   </NodeType>
   …
   <NodeType name="class">
   <Configuration threshold="0.6"/>
   <CompareFunctions>
   <CompareFunction
            weight="0.4"
            class="StringAttributeUsingLcsCS"
            parameter="name"/>
   <CompareFunction
            class="StringAttributeUsingEqualsCS"
            weight="0.05"
            parameter="visibility"/>
   <CompareFunction
            class="StringAttributeUsingEqualsCS"
            weight="0.05"
            parameter="isAbstract"/>
```

```
<CompareFunction
         binding="optional_true"
         class="OutgoingReferencesMatchedOrSimilarIO"
         weight="0.2"
         parameter="attrs"/>
<CompareFunction
         binding="optional_true"
         class="OutgoingReferencesMatchedOrSimilarIO"
         weight="0.2"
         parameter="methods"/>
<CompareFunction
         binding="optional_true"
         class="RemoteNodesMatchedOrSimilarIO"
         weight="0.05"
         parameter="I:generalization_src/O:generalization_tgt"/>
<CompareFunction
         binding="optional_true"
         class="RemoteNodesMatchedOrSimilarIO"
         weight="0.05"
         parameter="I:generalization_tgt/O:generalization_src"/>
</CompareFunctions>
</NodeType>
…
<NodeType name="operation">
<Configuration threshold="0.6"/>
<CompareFunctions>
<CompareFunction
         class="StringAttributeUsingLcsCS"
         weight="0.4" parameter="name"/>
<CompareFunction
         class="StringAttributeUsingEqualsCS"
         weight="0.04"
         parameter="visibility"/>
<CompareFunction
         class="StringAttributeUsingEqualsCS"
         weight="0.04"
         parameter="isAbstract"/>
<CompareFunction
         class="StringAttributeUsingEqualsCS"
         weight="0.04"
         parameter="ownerScope"/>
<CompareFunction
         binding="optional_true"
         class="OutgoingReferencesMatchedOrSimilarCO"
         parameter="parameters;0.25"
         weight="0.15"/>
<CompareFunction
         class="OutgoingReferenceMatchedOrSimilar"
         weight="0.15"
         parameter="returns"/>
<CompareFunction
         class="ParentsMatched"
         weight="0.15"/>
</CompareFunctions>
```

```
   …
   <NodeType name="parameter">
   <Configuration threshold="0.6"/>
   <CompareFunctions>
   <CompareFunction
            class="StringAttributeUsingLcsCS"
            weight="0.5"
            parameter="name"/>
   <CompareFunction
            class="OutgoingReferenceMatchedOrSimilar"
            weight="0.25"
            parameter="paramType"/>
   <CompareFunction
            class="ParentsMatched"
            weight="0.25"/>
   </CompareFunctions>
   </NodeType>
</SiDiffCompare>
```

B III: Simulink.compare.xml (Auszug)

```
   <SiDiffCompare>
   <Configurations>
      …
      <NodeType name="Line">
      <Configuration threshold="0.55" />
      <CompareFunctions>
      <If weight="1.0"
            condition="IsParentMatched"
            parameter="true">
      <Then>
            <CompareFunction
                  binding="optional_false"
                  class="OutgoingReferenceMatched"
                  weight="0.75"
                  parameter="toSourceBlock"/>
            <CompareFunction
                  binding="optional_false"
                  class="StringAttributeUsingEqualsCI"
                  weight="0.25"
                  parameter="SrcPort"/>
      </Then>
      </If>
      </CompareFunctions>
      </NodeType>
      …
      <NodeType name="Branch">
      <Configuration threshold="0.25" />
      <CompareFunctions>
            <If weight="1.0"
            condition="IsParentMatched"
            parameter="true">
```

```
   <Then>
   <CompareFunction
         binding="optional_false"
         class="OutgoingReferenceMatched"
         weight="0.75"
         parameter="toTargetBlock"/>
   <CompareFunction
         binding="optional_false"
         class="StringAttributeUsingEqualsCI"
         weight="0.25"
         parameter="DstPort"/>
   </Then>
   </If>
   </CompareFunctions>
   </NodeType>
   …
   <NodeType name="Port">
   <Configuration threshold="1.0" />
         <CompareFunctions>
         <If weight="1.0"
         condition="IsParentMatched"
         parameter="true">
   <Then>
   <CompareFunction
         binding="mandatory"
         class="StringAttributeUsingEqualsCI"
         weight="1.0"
         parameter="PortNumber"/>
   </Then>
   </If>
   </CompareFunctions>
   </NodeType>
…
<NodeType name="*">
<Configuration threshold="0.2" />
<CompareFunctions>
   <If weight="1.0"
         condition="IsParentMatched"
         parameter="true">
   <Then>
   <CompareFunction
               binding="mandatory"
               class="StringAttributeUsingEqualsCI"
               weight="0.2"
               parameter="Name"/>
   <CompareFunction
               binding="optional_false"
               class="OutgoingReferencesMatchedIO"
               weight="0.4"
               parameter="$nextBlock"/>
   <CompareFunction
               binding="optional_false"
               class="IncomingReferencesMatchedIO"
               weight="0.4"
```

```
                parameter="$nextBlock"/>
    </Then>
    </If>
    </CompareFunctions>
    </NodeType>

</Configurations>
</SiDiffCompare
```

Literaturverzeicnis

[[Ang05] ***Angermann, Anne, et al.*** Matlab - Simulink - Stateflow.
München: Oldenbourg Wissenschaftsverlag, 2005.

[Bla04] ***Black, Paul E.*** *Longest Common Substring.*
National Institute of Standards and Technology.
[Online] [Zitat vom: 2008. 11 15.]
http://nist.gov/dads/HTML/longestCommonSubstring.html.

[Bla06] ***Black, Paul E.*** *Longest Common Subsequence.*
National Institute of Standards and Technology.
[Online] [Zitat vom: 2008. 11 15.]
http://nist.gov/dads/HTML/longestCommonSubsequence.html.

[Bla08] ***Black, Paul E.*** *Levenshtein Distance.*
National Institute of Standards and Technology.
[Online] [Zitat vom: 15. 11 2008.]
http://www.nist.gov/dads/HTML/Levenshtein.html.

[Böc07] ***Böckenbauer, Hans Joachim und Bongartz, Dirk.*** Algorithmic Aspects Of Bioinformatics.
Berlin, Heidelberg, New York: Springer-Verlag, 2007.

[Bra00] ***Bramer, M.A.*** Knowledge Discovery and Datamining: theory and practice.
Hertfordshire: Institution of Engineering and Technology, 2000.

[Cor07] ***Cormen, Thomas, et al.*** Algorithmen - Eine Einführung.
München: Oldenbourg Wissenschaftsverlag GmbH, 2007.

[Day08] ***Daylight Inc.*** *Daylight Theory - Fingerprints.*
[Online] [Zitat vom: 15. 11 2008.]
http://www.daylight.com/dayhtml/doc/theory/theory.finger.html.

[For07] ***Forbig, Peter.*** Objektorientierte Softwareentwicklung mit UML.
München: Carl Hansa Verlag, 2007.

[Gad08] ***Gad, Roman.*** Testportal zur Unterstützung rechnerbasierter Experimente.
Universität Siegen: Diplomarbeit, 2008.

[Gas03] ***Gasteiger, Johann und Engel, Thomas.*** Chemoinformatics.
Weinheim: Wiley-VCH, 2003.

[Ham50] ***Hamming, R. W.*** Error Detecting And Error Correcting Codes.
s.l.: Bell Telephone Labratories, 1950.
[Online] [Zitat vom: 15. 11 2008.]
http://guest.engelschall.com/~sb/hamming/.

[Hit03] ***Hitz, Martin und Kappel, Gerti.*** UML@Work.
Heidelberg: dpunkt.verlag GmbH, 2003.

[Jon04] ***Jones, Neil C. und Pevzner, Pavel A.*** An Introduction to Bioinformatics Algorithms.
Massachusetts Institute of Technology: The MIT Press, 2004.

[Kel06] ***Kelter, Udo und Wenzel, Sven.*** Model-Driven Design Pattern Detection.
Benevento, Italien: Konferenzbeitrag, 2006.

[Koh06] ***Kohlbacher, Oliver.*** *Vorlesung Wirkstoffentwurf 2006/2007.*
[Online] [Zitat vom: 15. 11 2008.]
http://www-bs.informatik.uni-tuebingen.de/Teaching/Old/WS06/WE/Folien.

[Lan99] ***Lanza, Michele.*** Combining Metrics and Graphs for Object Oriented Reverse Engineering.
University of Bern, Switzerland: Master Thesis, 1999.

[Lea03] ***Leach, Andrew R. und Gillet, Valerie J.*** An introduction to chemoinformatics.
Berlin, Heidelberg, New York: Springer Verlag, 2003.

[Mas03] ***Masloch, Andrè.*** URLChecker - Ein Agent zur intelligenten, seiteninhaltsbasierten URL-Überprüfung.
Universität Dortmund: Diplomarbeit, 2003.

[Nee70] ***Needleman, Saul B. und Wunsch, Christian D.*** *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal Of Melecular Biology. *Vol. 48, 1970.*

[Pie05] ***Pietruszka, Wolf Dieter.*** MATLAB in der Ingenieurspraxis. *Wiesbaden: B. G. Teubner Verlag, 2005.*

[Rup05] ***Rupp, Chris und u.A.*** UML2 Glasklar. *München, Wien: Carl Hanser Verlag, 2005.*

[See00] ***Seemann, Jochen und von Gudenberg, Jürgen Wolff.*** Software-Entwurf in UML. *Berlin, Heidelberg, New York: Springer Verlag, 2000.*

[Sel04] ***Selzer, Paul M. und Marhöfer, Richard J., Rowher, Andreas.*** Angewandte Bioinformatik - Eine Einführung. *Berlin, Heidelberg, New York: Springer-Verlag, 2004.*

[Smi81] ***Smith, Temple und Waterman, Michael.*** *Identification of Common Molecular Subsequences.* Journal of Molecular Biologie. *Vol. 147, 1981.*

[Tre07] ***Treude, Christoph.*** Einsatz multidimensionaler Suchsstrukturen zur Optimierung der Bestimmung von Dokumentdifferenzen. *Universität Siegen: Diplomarbeit, 2007.*

[Weh04] ***Wehren, Jürgen.*** Ein XML-basiertes Differenzwerkzeug für UML-Diagramme. *Universität Siegen: Diplomarbeit, 2004.*

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen aus dem Internet.

Diejenigen Paragraphen der für mich gültigen Prüfungsordnung, welche etwaige Betrugsversuche betreffen, habe ich zur Kenntnis genommen.

Der Speicherung meiner Bachelor- (Master-, Diplom-) Arbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

_____ _____

(Ort, Datum) *(Unterschrift der Verfasserin/des Verfassers)*

Siegen, den

_____

PD DR. J. Ehlgen
(Leiter des Prüfungsamtes FB 5)