technische universität
dortmund

# Beschleunigung hierarchischer Clusterverfahren für allgemeine metrische Distanzmaße

Till Schäfer

Algorithm Engineering Report
**TR13-1-002**
Juni 2013
ISSN 1864-4503

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

technische universität
dortmund

Diplomarbeit

# Beschleunigung hierarchischer Clusterverfahren für allgemeine metrische Distanzmaße

**Till Schäfer**

**21. September 2012**

Betreuer:
Prof. Dr. Petra Mutzel
Dipl.-Inf. Nils Kriege

Fakultät für Informatik
Algorithm Engineering (LS11)
Technische Universität Dortmund
http://ls11-www.cs.tu-dortmund.de

# Inhaltsverzeichnis

# Kapitel 1.

# Einleitung

Diese Diplomarbeit beschäftigt sich mit der Beschleunigung von sequenziellen, agglomerativen, hierarchischen, nicht-überlappenden (SAHN) Clusterverfahren für allgemeine metrische Distanzmaße. Das Ziel besteht darin, die Performanz der Clusterverfahren in der Software Scaffold Hunter zu verbessern. Die Arbeit umfasst die Auswahl eines geeigneten Clusterverfahrens, die Auswahl einer passenden Technik zur Beschleunigung, dessen Implementierung in der Software *Scaffold Hunter* sowie eine theoretische und praktische Performanz- und Qualitätsanalyse.

## 1.1. Motivation und Hintergrund

Clustering ist ein Data-Mining-Verfahren zum Finden homogener (bzw. ähnlicher) Gruppen in Datenmengen. Es dient zur explorativen Analyse von Daten, für die keine Klassifikation bekannt ist oder bei denen bekannte Klassifikationen aus diversen Gründen nicht zu dem gewünschten Ergebnis führen. Da keine bekannte Klassifikation angelernt wird, sondern eine neue Klassifikation geschaffen wird, spricht man bei Clusterverfahren auch von unüberwachten Lernverfahren. Zur Anwendung kommt Clustering in einer Vielzahl verschiedener Fachrichtungen und Anwendungsbereiche. Beispielhaft lassen sich die Fachrichtungen Informatik, Medizin, Molekularbiologie und Soziologie sowie die Anwendungsgebiete Mustererkennung, Storage and Retrieval, Netzwerkanalyse (u. a. soziale Netzwerke) und Pharmazeutischer Strukturfindungsprozess nennen (s. a. [MC12; DB03]). Clustering kann unter anderem dazu verwendet werden, Daten zu strukturieren bzw. zu klassifizieren, relevante Teilmengen der Daten zu finden, ein Sampling mit großer Diversität zu erstellen oder Daten grafisch aufzubereiten. Die Bandbreite der Anwendungsmöglichkeiten ist sehr groß und der Bedarf an schnellen Clusterverfahren wächst mit der ständig zunehmenden Speicherung großer Datenmengen.

Die Software Scaffold Hunter ist ein Werkzeug zur intuitiven Visualisierung und explorativen Analyse des chemischen Strukturraumes. Seit Version 2.0 bietet sie die Möglichkeit, Moleküldatenbanken zu clustern. Zum Einsatz kommt ein SAHN-Clusterverfahren mit dem Namen NNChain (s. a. [Mur85], Abschnitt 4.1.3.1). Visualisiert durch ein Dendrogramm

bietet es dem Benutzer die Möglichkeit, den chemischen Strukturraum (d. h. die Menge aller Moleküle) zu gruppieren und damit die Informationsflut auf ein erfassbares Maß zu reduzieren, einen Einblick in die zugrundeliegende Ähnlichkeitsstruktur der Daten zu erhalten und eine Auswahl von – für den Benutzer relevanten – Molekülen zu finden. In der Praxis ist die Performanz des Verfahrens jedoch nur für kleine bis mittelgroße Moleküldatenbanken und wenig rechenaufwändige Distanzmaße ausreichend. Die Größe öffentlicher Moleküldatenbanken mit mehreren Millionen Molekülen (z. B. PubChem[1], ZINC[2]) übersteigt diese Grenze der Anwendbarkeit bei weitem. Daher wird sich diese Arbeit mit der Suche nach alternativen, schnelleren Clusterverfahren auseinandersetzen.

## 1.2. Aufbau der Arbeit

Kapitel 2 beschäftigt sich mit den Grundlagen des Clusterings und gibt einen Überblick über Klassen von Clusterverfahren, Distanzmaße, Linkage-Verfahren und einen Einblick in mögliche Visualisierungen der Ergebnisse. Zudem wird die Software Scaffold Hunter vorgestellt und bestehende Programmstrukturen erläutert, in denen der zu implementierende Algorithmus eingebettet wird.

Eine Analyse der Problemstellung wird in Kapitel 3 vorgenommen. Hierbei werden Die Rahmenbedingungen, die ein beschleunigtes hierarchischen Clusterverfahren erfüllen muss, herausgearbeitet.

In Kapitel 4 werden bekannte Clusterverfahren (dichtebasierte Verfahren, gitterbasierte Verfahren, SAHN-Verfahren) und bekannte Ansätze zur Beschleunigung (Datenverdichtung, Indexstrukturen, Pivot-Ansatz) allgemeiner Clusterverfahren vorgestellt, sowie auf ihre Eignung in Bezug auf die Problemstellung untersucht.

Schließlich wird in Kapitel 5, auf Grundlage der vorgestellten Verfahren aus Kapitel 4, ein neues heuristisches SAHN-Clusterverfahren entwickelt, welches den in Kapitel 3 erarbeiteten Anforderungen, soweit wie möglich gerecht wird. In diesem Kapitel wird daher die Auswahl eines Beschleunigungs- und Clusterverfahrens begründet. Anschließend wird die Kombination dieser Verfahren umgesetzt und notwendige Anpassungen an den bestehenden Verfahren beschrieben. Des Weiteren wird die Implementierung des neuen Verfahrens in der Software Scaffold Hunter erläutert.

Kapitel 6 umfasst eine experimentelle Performanz- und Qualitätsanalyse des heuristischen SAHN-Clusterverfahrens. Ein weiteres Ziel dieses Kapitels besteht in der Suche nach Richtwerten für die richtige Wahl der Parameter des Verfahrens. Um diese Ziele zu erreichen wird zunächst eine Einführung in verschiedene Qualitätsmaße gegeben, die Implementierung eines Evaluationsframeworks beschrieben und das Testdesign erläutert. Anschließend werden die Ergebnisse vorgestellt und analysiert.

[1] http://pubchem.ncbi.nlm.nih.gov/
[2] http://zinc.docking.org/

Das Kapitel 7 fasst die wichtigsten Erkenntnisse und Ergebnisse dieser Arbeit zusammen. Darauf aufbauend, wird ein Ausblick über mögliche Verbesserungen und noch zu untersuchende Probleme des Verfahrens gegeben.

# Kapitel 2.

# Grundlegendes

Dieses Kapitel beschreibt die Grundlagen der Clusteranalyse und stellt die Software Scaffold Hunter vor. Es werden für den weiteren Verlauf der Arbeit benötigte, grundlegende Notationen und Begrifflichkeiten definiert.

## 2.1. Clustering

Clustering (auch Clusteranalyse) ist ein Verfahren zum Finden homogener Gruppen in Datenmengen. Leider ist in der wissenschaftlichen Literatur keine allgemeingültige Definition des Clusterings zu finden. Alle mir bekannten Definitionen beziehen sich auf spezielle Klassen von Clusterverfahren. Daher sei im Folgenden eine eigene Formalisierung des Problems (Definition 2.1.1) gegeben, welche mit allen in dieser Diplomarbeit vorgestellten Clusterverfahren kompatibel ist.

**2.1.1 Definition (Clustering).** Sei $\mathcal{X} = \{x_1, \ldots, x_n\}$ eine Menge von Objekten und $q : \mathcal{P}(\mathcal{P}(\mathcal{X})) \to \mathbb{R}$ eine Qualitätsfunktion. Dann ist die Aufgabe des Clustering die Menge von Clustern $\mathcal{C} = \{C_1, \ldots, C_k\} \in \mathcal{P}(\mathcal{P}(\mathcal{X}))$ zu finden, so dass die Qualitätsfunktion $q(\mathcal{C})$ optimiert wird. Die Qualitätsfunktion stellt daher eine Abbildung des Clustering auf die Qualität dar. Sie wird immer mit Hilfe einer Distanzfunktion (auch Distanzmaß) $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ definiert.

Im Gegensatz zum überwachten Lernen geht es beim Clustering also nicht darum die Relation zwischen den Merkmalen der Beobachtungen und einer bekannten Klassifikation zu lernen, sondern darum Aspekte der zugrunde liegenden Verteilung der Daten zu erkennen. Deshalb spricht man auch von unüberwachtem Lernen [DB03].

Der Zusammenhang zwischen Distanz- und Qualitätsfunktion ist nicht selbsterklärend. Wie dieser Zusammenhang modelliert wird ist genau der Kernbestandteil eines Clusterverfahrens. Er definiert welche Cluster-Arten oder Formen entstehen. Im folgenden sei ein Beispiel gegeben, welches deutlich macht wie dieser Zusammenhang modelliert werden kann. Ein Clusterverfahren kann versuchen die Intra-Cluster-Varianz zu minimieren. Dies führt zu möglichst kompakten, in der Ausdehnung begrenzten Clustern. Die Varianz

wiederum kann mit Hilfe der Distanzen aller Objekte innerhalb eines Clusters berechnet werden. Welche Distanzfunktion gewählt wird, ist in diesem Fall unabhängig vom generellen Ziel des Verfahrens (Minimierung der Varianz). In vielen Verfahren ist die Distanzfunktion daher ein Parameter des Clusterverfahrens und kann dazu verwendet werden das Clusterverfahren an die gegebenen Anforderungen anzupassen.

Einige Clusterverfahren, wie z. B. k-Means [Elk03], setzen die Clusteranzahl $k$ als Parameter voraus. Andere Verfahren finden die Anzahl der Cluster anhand von Schwellenwerten oder andern Parametern des Modells. Zum Beispiel benötigt DBSCAN [EKS+96] lediglich einen Radius $\epsilon$ und einen Schwellenwert *MinPts* als Parameter (vgl. Abschnitt 4.1.1.1).

### 2.1.1. Klassen von Clusterverfahren

Clusterverfahren können aufgrund vieler Kategorien klassifiziert werden. Es folgt eine Auswahl von für diese Diplomarbeit relevanten Klassen.

#### 2.1.1.1. Überlappendes Clustering

Bei überlappenden Clusterverfahren kann ein Objekt $x \in \mathcal{X}$ Element von mehreren Clustern sein. Kann $x$ nur in einem einzigen Cluster vorkommen, spricht man von nichtüberlappenden Clusterverfahren.

#### 2.1.1.2. Fuzzy-Clustering

Beim Fuzzy Clustering wird die Zugehörigkeit einzelner Objekte $x \in \mathcal{X}$ zu den Clustern $C$ nur mit einem Anteil bzw. einer Wahrscheinlichkeit angegeben. Ein Objekt $x$ könnte z. B. mit einer Wahrscheinlichkeit von 0,4 in Cluster $C_i$ und mit einer Wahrscheinlichkeit von 0,6 in $C_j$ $(i \neq j)$ liegen.

#### 2.1.1.3. Graphbasiertes Clustering

Graph Clustering erhält als Eingabe anstatt einer Menge von Objekten $\mathcal{X}$ und einem Distanzmaß $d$ einen Graphen $G = (V, E)$. Geclustert werden die Knoten $V$ des Graphen. Die Kanten definieren die Ähnlichkeit oder Konnektivität zwischen den Knoten und sind damit das Gegenstück zum Distanzmaß aus Definition 2.1.1 [Sch07]. Graph Clustering und Clustering gemäß Theorem 2.1.1 sind eng miteinander verwandt und die Probleme können teilweise ineinander überführt werden. Als Beispiel sei Chameleon genannt [KHK99]: In diesem Verfahren wird aus $\mathcal{X}$ und $d$ ein $k$-NN-Graph ($k$-nächster-Nachbar-Graph) konstruiert. Jeder Knoten $v \in V$ entspricht dabei einem Objekt aus $\mathcal{X}$. Eine Kante zwischen zwei Knoten $v$ und $w$ wird genau dann eingefügt, wenn $w$ in der $k$-nächsten Nachbarschaft von $v$ und $v$ in der $k$-nächsten Nachbarschaft von $w$ liegt. Das Clustering wird dann auf dem konstruierten Graphen ausgeführt.

#### 2.1.1.4. Hierarchisches Clustering

Hierarchische Clusterverfahren beschreiben eine hierarchische Verschmelzung oder Aufteilung der Daten (agglomerative bzw. diversive Verfahren). Bei der agglomerativen Variante startet das Clusterverfahren mit einelementigen Clustern (Singletons) und verschmilzt sukzessiv die beiden Cluster mit der kleinsten Distanz bis nur noch ein Cluster übrig bleibt [DB03]. Bei der diversiven Variante ist es genau umgekehrt. Die ausgegebene Menge von Clustern $\mathcal{C}$ ist daher nicht *flach*, sondern durch ihre Verschmelzung bzw. Aufteilung strukturiert.

Das Ergebnis eines hierarchischen Clusterverfahrens lässt sich daher als gewurzelter Baum beschreiben. Die Knoten dieses Baumes entsprechen Clustern. Wenn ein Cluster $C$ im Dendrogramm die Kinder $C_1, \ldots, C_k$ besitzt, dann entspricht die Vereinigung der Cluster $C_1, \ldots, C_k$ dem Cluster $C$. Die Wurzel des Baumes entspricht daher dem Cluster, welcher die gesamte Menge $\mathcal{X}$ umfasst und die Blätter entsprechen den Singletons. Häufig werden diese Ergebnisse als Dendrogramm dargestellt (s. a. Abbildung 2.1). In einem Dendrogramm repräsentiert die Höhe der Knoten die Distanz der verschmolzenen Cluster. Ein flaches (nicht hierarchisches) Clustering kann durch einen Schwellenwert der Distanz ermittelt werden, indem das Dendrogramm auf der entsprechenden Höhe horizontal geschnitten wird.

Bekannte Beispiele für hierarchische Clusterverfahren sind SAHN (s. a. Abschnitt 2.1.1.5), OPTICS [ABK+99] und CURE [GRS98].

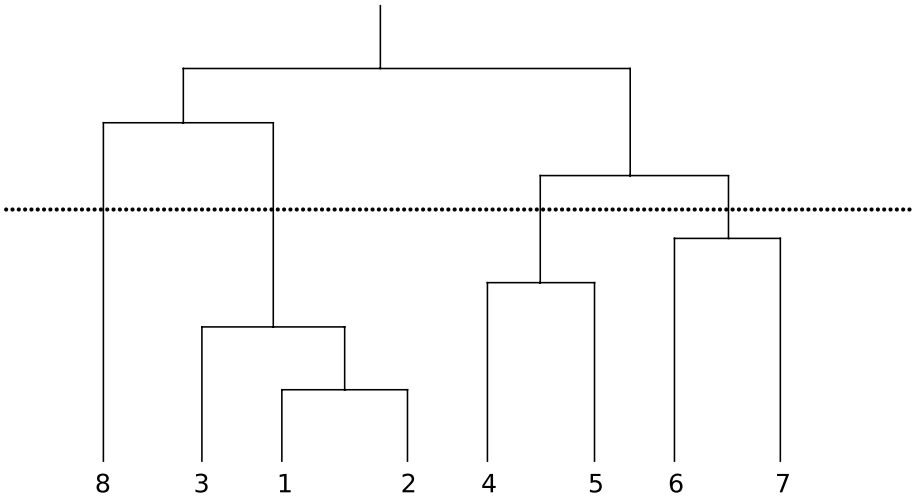



**Abbildung 2.1.:** Dendrogramm, Quelle: [DB03]

Der Vorteil eines hierarchischen Verfahrens liegt darin, dass der Benutzer die Clusteranzahl im Nachhinein beliebig variieren kann. Häufig gibt es keine natürliche Clusteranzahl, die den Daten zugrunde liegt. Ein Cluster kann beispielsweise intern erneut in kleinere Cluster zerfallen und damit existieren alternative, flache Clusterings, welche sich qualitativ nicht unterscheiden müssen. Es wäre in diesem Fall falsch die Clusteranzahl durch

einen Parameter des Verfahrens zu fixieren. Ein hierarchisches Clustering kann beide alternativen in den Ergebnissen kodieren und vermittelt daher einen tieferen Einblick in die Ähnlichkeitsstruktur der Daten als es ein flaches vermag. Diesem hohen Informationsgehalt des Ergebnisses steht in der Regel aber ein hoher Berechnungsaufwand gegenüber.

#### 2.1.1.5. SAHN-Clustering

Sequenzielle agglomerative hierarchische nicht-überlappende (SAHN) Clusterverfahren sind eine spezielle Klasse der hierarchischen Clusterverfahren. Wie der Name vermuten lässt, werden iterativ die beiden Cluster mit der geringsten Distanz verschmolzen (vgl. Algorithmus 2.1).

1: $AktiveCluster \leftarrow Singletons$
2: **while** $|AktiveCluster| > 1$ **do**
3: Finde Cluster $C_i$ und $C_j$ aus $AktiveCluster$ mit minimaler Distanz $D$
4: Verschmelze $C_i$ und $C_j$ zu $C_{i \cup j}$
5: $AktiveCluster \leftarrow (AktiveCluster \setminus \{C_i, C_j\}) \cup C_{i \cup j}$
6: **end while**

**Algorithmus 2.1:** Naives SAHN-Clustering

**Linkage-Verfahren** Bei SAHN-Clusterverfahren ist zwischen Distanzmaßen und dem Linkage-Verfahren zu unterscheiden. Das Distanzmaß $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ definiert die Distanz zwischen Objekten und das Linkage-Verfahren $D : \mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ entspricht der Inter-Cluster-Distanz. Letzteres verwendet die paarweisen Distanzen der Clusterelemente als Grundlage. Tabelle 2.1 gibt einen Überblick über die bekanntesten Linkage-Verfahren.

**Tabelle 2.1.:** Linkage-Verfahren

| Linkage | Definition | Bemerkung |
|---|---|---|
| Single-Linkage | $D(C_i, C_j) = \min\{d(x,y) \mid x \in C_i \wedge y \in C_j\}$ | |
| Complete-Linkage | $D(C_i, C_j) = \max\{d(x,y) \mid x \in C_i \wedge y \in C_j\}$ | |
| Average-Linkage | $D(C_i, C_j) = \sum_{x \in C_i, y \in C_j} \left( \frac{d(x,y)}{\lvert C_i \rvert \ \lvert C_j \rvert} \right)$ | |
| Ward-Linkage | $D(C_i, C_j) = \sigma^2(C_i \cup C_j) - \sigma^2(C_i) - \sigma^2(C_i)$ | $\sigma^2$ = Varianz |
| Zentroid-Linkage | $D(C_i, C_j) = d(\bar{C}_i, \bar{C}_j)$ | $\bar{C}_x$ = Zentroid |
| Median-Linkage | $D(C_i, C_j) = d(\tilde{C}_i, \tilde{C}_j)$ | $\tilde{C}_x$ = Medoid |

Für Median- und Zentroid-Linkage sind die Definitionen für den Zentroid und den Medoit [SHR97] notwendig um ihre Funktionsweise zu verstehen:

**2.1.2 Definition (Zentroid).** Ein Zentroid ist der Mittelpunkt aller Objekte eines Clusters im euklidischen Vektorraum:

$$\bar{C} = \frac{1}{|C|} \sum_{\vec{x} \in C} \vec{x} \tag{2.1}$$

**2.1.3 Definition (Medoit).** Ein Medoit ist das Objekt mit der geringsten durchschnittlichen Distanz zu allen anderen Objekten eines Clusters:

$$\tilde{C} = \text{argmin}_x \left( \sum_{y \in C \setminus \{x\}} d(x, y) \right) \tag{2.2}$$

Es ist zu beachten, dass Ward- und Zentroid-Linkage für das euklidische Distanzmaß (s. a. Abschnitt 2.1.2) definiert sind. Median-Linkage ist zwar durch die Definition des Medoit nicht auf das euklidische Distanzmaß beschränkt, kann aber in der Praxis nur für dieses effizient eingesetzt werden. Die Gründe liegen in der Lance-Williams-Update-Formel (s. u.) begründet, welche Median-Linkage ausschließlich in Kombination mit dem euklidischen Distanzmaß beherrscht. Ward- Median- und Zentroid-Linkage finden jedoch auch in Kombination mit anderen Distanzmaßen und der Lance-Williams-Update-Formel (für das euklidische Distanzmaß) eine breite Anwendung, da ihre wesentlichen Eigenschaften erhalten bleiben. Für Ward-Linkage gibt es zudem eine wahrscheinlichkeitstheoretische Begründung, weshalb dieses Verfahren auch auf andere Distanzmaße angewandt werden kann (s. a. [Bat88]).

**Laufzeit** Die Laufzeit für das SAHN-Clustering für allgemeine Linkage-Verfahren und Distanzmaße, die in konstanter Zeit berechnet werden können, beträgt $\Theta(n^2)$ (Beweis: [Mü11]). Diese Schranke gilt auch für die Anzahl der Distanzberechnungen.

**Lance-Williams-Update-Formel** Die Lance-Williams-Update-Formel dient zur inkrementellen Berechnung der Inter-Cluster Distanzen (s. a. [LW67; MC12]). Sie ermöglicht die Berechnung der Distanz eines beliebigen Clusters $C_k$ zu einem Cluster $C_{i \cup j}$, der aus der Verschmelzung der Cluster $C_i$ und $C_j$ entstanden ist, in konstanter Zeit. Die Lance-Williams-Update-Formel lautet:

$$D(C_{i \cup j}, C_k) = \alpha_i \; D(C_i, C_k) + \alpha_j \; D(C_j, C_k) + \beta \; D(C_i, C_j) + \gamma \; |D(C_i, C_k) - D(C_j, C_k)| \tag{2.3}$$

Es ist zu beachten, dass die Lance-Williams-Update-Formel von dem verwendeten Linkage-Verfahren abhängig ist. Diese Abhängigkeit kann über die Parameter $\alpha$, $\beta$ und $\gamma$ realisiert werden. Die entsprechenden Werte für ein konkretes Linkage-Verfahren können Tabelle 2.2 entnommen werden. Die Parameter $\alpha_i$ und $\alpha_j$ sind symmetrisch in $i$ und $j$.

Die Lance-Williams-Update-Formel berechnet für Median- und Zentroid-Linkage nur dann die korrekten Distanzen, wenn auf beiden Seiten der Formel quadrierte Distanzen $D^2$ verwendet werden.

**Tabelle 2.2.:** Lance-Williams-Update-Formel, Quelle: [MC12]

| Linkage | $\alpha_i$ | $\beta$ | $\gamma$ | Bemerkung |
|---|---|---|---|---|
| Single-Linkage | 0,5 | 0 | $-0{,}5$ | |
| Complete-Linkage | 0,5 | 0 | 0,5 | |
| Average-Linkage | $\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$ | 0 | 0 | |
| Ward-Linkage | $\frac{\lvert C_i\rvert+\lvert C_k\rvert}{\lvert C_i\rvert+\lvert C_j\rvert+\lvert C_k\rvert}$ | $\frac{C_k\rvert}{\lvert C_i\rvert+\lvert C_j\rvert+\lvert C_k\rvert}$ | 0 | |
| Zentroid-Linkage | $\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$ | $\frac{\lvert C_i\rvert}{(\lvert C_i\rvert+\lvert C_j\rvert)^2}$ | 0 | quadrierte Distanz |
| Median-Linkage | 0,5 | $-0{,}25$ | 0 | quadrierte Distanz |

### 2.1.2. Distanzmaße

Das Distanzmaß $d$ aus Definition 2.1.1 gibt an, wie unähnlich sich zwei Objekte aus der Grundmenge $\mathcal{X}$ sind. Im Allgemeinen gilt, je größer der Wert desto größer die Unähnlichkeit. Eine Distanz verhält sich damit antiproportional zu der Ähnlichkeit $Sim$ der zu vergleichenden Objekte. Für 1-normierte Distanzen gilt:

$$Sim(x,y) = 1 - d(x,y) \tag{2.4}$$

Distanzmaße stellen eine einfach zu modifizierende Komponente vieler Clusterverfahren dar. Häufig stellt die Wahl des Distanzmaßes (in Kombination mit den verwendeten Merkmalen bzw. Attributen) den ersten Schritt dar, um zu spezifizieren an welchen Eigenschaften eines Objektes der Benutzer interessiert ist. Auf Grund dessen, wurden eine Vielzahl verschiedener Distanzmaße vorgestellt, welche für die unterschiedlichsten Problemstellungen optimiert wurden.

#### 2.1.2.1. Häufig verwendete Distanzmaße

Dieser Abschnitt stellt häufig verwendete und für diese Arbeit relevante Distanzmaße vor. Es wird kein Anspruch auf Vollständigkeit erhoben. Dies ist aufgrund der Vielzahl von bekannten Distanzmaßen, im Rahmen dieser Diplomarbeit auch nicht möglich. Die vorgestellten Distanzmaße sind allesamt generischer Natur und unabhängig von einer konkreten Anwendung entworfen worden.

**p-Norm** Mit Hilfe der $p$-Norm kann die Länge von Vektoren berechnet werden. Sie ist folgendermaßen definiert (s. a. [MC12]):

$$||\vec{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}, p \in \mathbb{Z}^+ \tag{2.5}$$

Je nach gewähltem Parameter $p$ wird die $p$-Norm auch als Betragssummennorm ($p = 1$), Euklidische Norm ($p = 2$) oder Tschebyschow Norm ($p = \infty$) bezeichnet. Der Wertebereich

der $p$-Norm ist $[0, \infty[$. Die $p$-Norm ist genau dann 0 wenn $\vec{x} = 0$. Die Distanz zwischen zwei Koordinaten $\vec{x}_1$ und $\vec{x}_2$ lässt sich berechnen, indem die $p$-Norm für $\vec{x} = \vec{x}_1 - \vec{x}_2$ berechnet wird.

**Jaccard Koeffizient** Der Jaccard Koeffizient (s. a. [FVB02]) misst den Anteil des Schnittes zweier Mengen an der Vereinigungsmenge:

$$Jaccard(A, B) = \frac{A \cap B}{A \cup B} \tag{2.6}$$

Der Wertebereich des Jaccard Koeffizienten ist [0,1]. Er nimmt genau dann den Wert 1 an, wenn die beiden Mengen identisch sind, und 0, wenn die beiden Mengen keine gemeinsamen Elemente beinhalten. Der Jaccard Koeffizient ist also ein 1-normiertes Ähnlichkeitsmaß. Die Jaccard-Metrik (s. a. Abschnitt 2.1.2.2) ist ein Distanzmaß und ist definiert als:

$$JaccardMetrik(A, B) = 1 - Jaccard(A, B) \tag{2.7}$$

**Tanimoto Koeffizient** Der Tanimoto Koeffizient (s. a. [FVB02]) ist vom Jaccard Koeffizienten abgeleitet und eignet sich als Ähnlichkeitsmaß zwischen zwei Bit-Fingerprints $F_1$ und $F_2$. Ein Bit-Fingerprint ist als Vektor von 1/0 Werten zu verstehen.
Im Folgenden sei:

- $N_{11}$ die Anzahl der gemeinsamen 1-Bits von $F_1$ und $F_2$
- $N_{10}$ die Anzahl der 1-Bits von $F_1$ die nicht in $F_2$ gesetzt sind
- $N_{01}$ die Anzahl der 1-Bits von $F_2$ die nicht in $F_1$ gesetzt sind

Der Tanimoto Koeffizient ist dann definiert als:

$$Tanimoto(F_1, F_2) = \frac{N_{11}}{N_{11} + N_{10} + N_{01}} \tag{2.8}$$

Wenn man $N_{11}$ als Schnitt der 1-Bits und $N_{11} + N_{10} + N_{01}$ als Vereinigung betrachtet, wird der Zusammenhang zu Jaccard deutlich. Die Tanimoto Metrik (s. a. Abschnitt 2.1.2.2) lässt sich analog zur Jaccard Metrik berechnen.

#### 2.1.2.2. Metrik

In Bezug auf den Rechenaufwand von Clusterverfahren hat sich eine Einschränkung auf metrische Distanzfunktionen häufig als hilfreich erwiesen (vgl. Abschnitt 4.2.3).

**2.1.4 Definition (Metrik).** Ein metrisches Distanzmaß erfüllt die folgenden drei Bedingungen [Kel75]:

- Symmetrie: $d(x, y) = d(y, x)$

- Dreiecksungleichung: $d(x, y) + d(y, z) \geq d(x, z)$
- $d(x, y) = 0 \Leftrightarrow x = y$

Häufig wird zusätzlich die Bedingung $d(x, y) \geq 0$ genannt. Sie folgt jedoch aus den drei oben genannten Bedingungen und ist damit redundant.

Alle in Abschnitt 2.1.2.1 vorgestellten Distanzmaße (s. a. [Lip99; GL86]) beschränken sich auf den metrischen Raum. Damit gilt diese Einschränkung ebenfalls für die in der geometrischen Realität geltende euklidische Distanz und wird daher häufig als intuitive oder natürliche Einschränkung angesehen. Dennoch gibt es einige Beispiele in der realen Welt die nicht metrischer Natur sind. Die Distanzen in einem Straßennetz mit Einbahnstraßen sind z. B. nicht symmetrisch.

## 2.2. Scaffold Hunter

Das Ziel des Scaffold Hunter Projektes[1] ist es, für Chemiker und Molekularbiologen ein Werkzeug bereitzustellen, mit dessen Hilfe die Medikamenten- bzw. Molekülsuche vereinfacht werden kann. Allein die Anzahl von geschätzten $10^{160}$ unterschiedlichen synthetisierbaren Molekülen und $10^{60}$ Wirkstoffen macht deutlich, dass die Untersuchung jedes einzelnen Moleküls auf seine medizinische Wirkung nicht möglich ist, sondern ein zielgerichtetes Vorgehen erfordert. Die Software muss dem Wissenschaftler helfen die Daten zu strukturieren und die Informationsdichte auf ein erfassbares Maß zu reduzieren. Leider gibt es keinen bekannten automatisierbaren Prozess, um von einer Krankheit auf einen Wirkstoff für ein Medikament schließen zu können. Die Zusammenhänge müssen auf kreative Art und Weise immer wieder neu entdeckt werden. Dies macht eine vollautomatische Auswertung der Daten unmöglich. Die Software Scaffold Hunter ist daher ein Werkzeug zur Visualisierung und explorativen Analyse des chemischen Strukturraumes. Sie soll einen intuitiven Zugang zu den Daten ermöglichen und helfen neue Erkenntnisse zu generieren und festzuhalten.

Der Scaffold Hunter ist in der Programmiersprache Java geschrieben und wurde unter der Open-Source Lizenz GPLv3[2] veröffentlicht. Zum Zeitpunkt der Anfertigung dieser Diplomarbeit trug das letzte offizielle Release die Versionsnummer 2.0. Der Scaffold Hunter ging im Rahmen der Projektgruppe 552 unter Leitung von Karsten Klein, Nils Kriege und Carsten Gutwenger aus der Version 1.7 hervor. Initiiert wurde das Projekt von Karsten Klein und Stefan Wetzel im Rahmen der Projektgruppe 504. Beide Versionen sind in Kooperation mit dem Max-Planck-Institut für molekulare Physiologie entstanden.

In der Version 1 umfasste die Funktionalität vor allem die Darstellung chemischer Datenbanken mit Hilfe des Scaffold-Prinzips (s. a. [SER+07]). Das Scaffold-Prinzip beschreibt ein

[1] http://sourceforge.net/projects/scaffoldhunter/
[2] https://www.gnu.org/licenses/gpl.html

Verfahren zur Klassifikation von Molekülen anhand ihres Rumpfes. Durch sukzessives Entfernen von Seitenketten und Ringen erhält man immer kleinere Moleküle, welche Scaffold genannt werden. Jeder Scaffold dient damit als Identifikator für eine Menge von größeren Molekülen und/oder Scaffolds. Die aus dem Scaffold-Prinzip gewonnene Baumstruktur wird im Scaffold Hunter kreisförmig dargestellt (vgl. Abbildung 2.2) und kann zusammen mit visuellen Hervorhebungen (Farbe, Größe, etc. von Kanten und Knoten) und benutzerdefinierten Annotationen Zusammenhänge in den Daten sichtbar machen. Zudem kann der Datensatz nach Attributen und mit Hilfe einer Substruktursuche gefiltert werden (s. a. [WKR+09]).

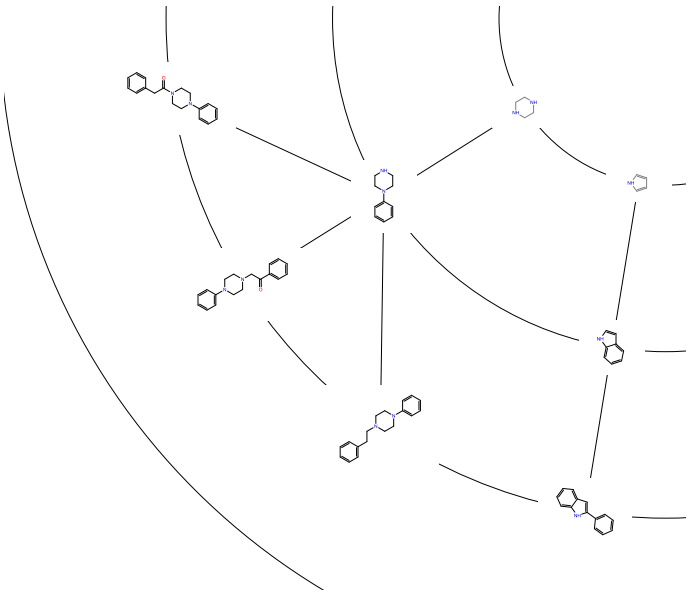


**Abbildung 2.2.:** Ausschnitt Scaffoldbaum

Die Version 2.0 des Scaffold Hunter wurde zu großen Teilen neu entworfen und neu programmiert sowie durch ein Ansichten-Konzept ergänzt. Die Scaffoldbaumansicht wurde durch eine Tabellen-, eine Plot- und eine Dendrogrammansicht ergänzt. Die Selektion von Molekülen ist über alle Ansichten synchronisiert und macht es damit besonders leicht, ausgewählte Moleküle in verschiedenen Ansichten zu betrachten.

Im Rahmen dieser Diplomarbeit ist die Dendrogrammansicht (Abbildung 2.3) von besonderer Bedeutung. Mit ihr wurde eine zum Scaffold-Prinzip alternative Klassifikationsmöglichkeit (SAHN-Clustering) eingeführt. Es ist möglich unterschiedliche Distanzmaße, Linkage-Verfahren und Attribute für das Clustering auszuwählen. Mit Hilfe eines wählbaren Schwellenwertes kann die konkrete Clusteranzahl interaktiv und ohne Neuberechnung des Clustering gewählt werden. Einzelne Teilbäume des Dendrogramms können der Selektion hinzugefügt oder aus ihr entfernt werden. Dadurch kann der Benutzer die Ähnlichkeitsstruktur der Daten schnell und zielführend analysieren.

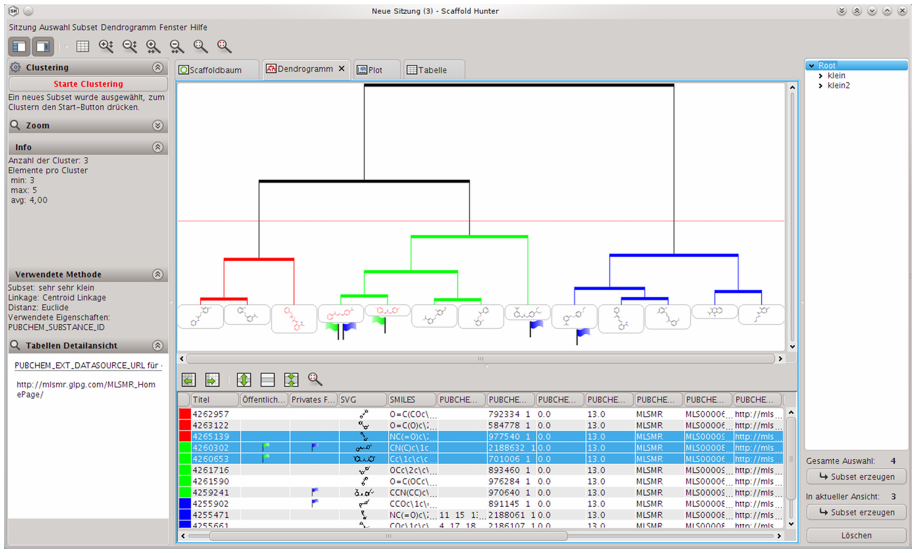



**Abbildung 2.3.:** Dendrogrammansicht

Ein neues Teilmengenkonzept erlaubt seit Version 2.0 das Speichern von Teilmengen der Daten in einer Baumstruktur. Dadurch kann der Benutzer nachvollziehen, durch welche Schritte bzw. Filterprozesse er zu einer Auswahl an relevanten Molekülen gelangt ist. Jede Teilmenge kann in vorhandenen oder neuen Ansichten angezeigt werden. Die Ansichten können nebeneinander oder übereinander im selben Fenster (Split-Screen-Konzept) oder in mehreren Fenstern angezeigt werden. Verschiedene Arbeitsschritte und Ansichten können so leicht miteinander verglichen und kombiniert werden.

Mögliche Datenquellen für den Scaffold Hunter sind verschiedene Dateiformate (SDF, CSV) und relationale Datenbanken. Eigenschaften (Attribute) der Moleküle können nicht nur importiert, sondern auch berechnet werden. Über diesen Weg können z. B. aus der Molekülstruktur Fingerprints für das Clustering generiert werden. Pluginschnittstellen für Import-Filter und zur Berechnung von Attributen tragen zur einfachen und flexiblen Erweiterbarkeit des Scaffold Hunter bei.

# Kapitel 3.

# Analyse der Problemstellung

Diese Kapitel beschäftigt sich mit der Analyse des Laufzeitverhaltens des NNChain-Algorithmus und leitet die Anforderungen an ein alternatives Clusterverfahren aus dieser Analyse und den Anwendungsfällen des Clustering im Scaffold Hunter ab.

## 3.1. Das Laufzeitverhalten des NNChain-Algorithmus

Ein großes Problem der bisherigen Implementierung des Clusterings im Scaffold Hunter ist die benötigte Laufzeit. Der NNChain-Algorithmus (s. a. Abschnitt 4.1.3.1) besitzt sowohl in der Theorie, als auch in der Praxis eine quadratische Laufzeit. Abhängig von dem verwendeten Linkage-Verfahren und dem verwendeten Distanzmaß benötigt der NNChain-Algorithmus eine Distanzmatrix oder Cluster-Zentroiden zum Ermitteln der Inter-Cluster-Distanz $D$. Die Distanzmatrix enthält zu Beginn die paarweisen Distanzen aller Singleton-Cluster und belegt damit quadratisch viel Speicherplatz. Der repräsentantenbasierte NNChain-Algorithmus kommt mit linear viel Speicher aus. Repräsentanten können jedoch nur bei Verwendung von Ward-Linkage in Kombination mit dem Euklidischen Distanzmaß eingesetzt werden und stellen damit einen Sonderfall dar. Abbildung 3.1 und Abbildung 3.2 veranschaulichen die experimentelle Laufzeit des distanzmatrixbasierten NNChain-Algorithmus. Die Laufzeit teilt sich daher in zwei Teile auf. Zum einen wird die Laufzeit der Initialisierung der Distanzmatrix und zum anderen die Laufzeit des NNChain-Algorithmus selbst gemessen. Die Messungen wurden auf einer Intel Core2 Quad CPU Q9450 mit 2.66GHz und einem Thread durchgeführt.

Wenn man den Scaffold Hunter als interaktives Werkzeug auffasst, dann ist eine Laufzeit von mehr als einigen zehn Minuten für die Berechnung des Clustering nicht praktikabel. Der NNChain-Clustering-Algorithmus kann dieser Anforderung für Datensätze jenseits einer Datensatzgröße von einigen zehntausend Objekten auf aktueller Hardware nicht gerecht werden. Im Kontrast dazu beinhaltet die freie Moleküldatenbank PubChem zum aktuellen Zeitpunkt etwa 100 Millionen Substanzen[1]. In praktikabler Zeit ist daher nur ein Bruchteil der Datenbank zu clustern.

[1] http://www.ncbi.nlm.nih.gov/sites/entrez?term=all[filt]&cmd=search&db=pcsubstance

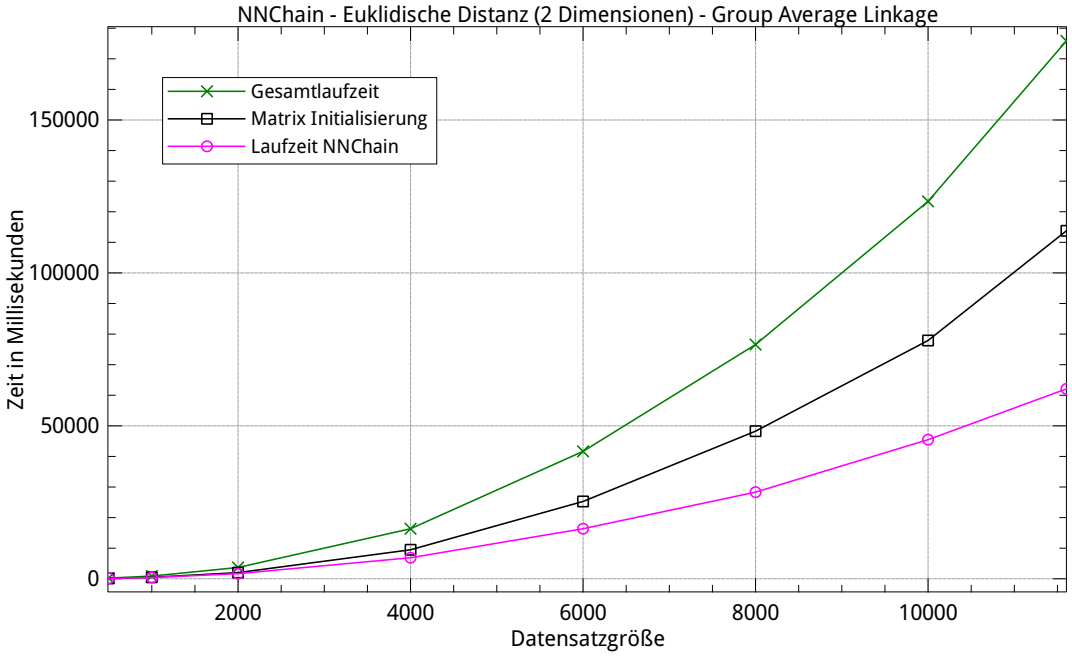



**Abbildung 3.1.:** Laufzeit NNChain, Euklidische Distanz, 2 Dimensionen, Average-Linkage

Hinzu kommt, dass für die Berechnung der verwendeten Distanzmatrix quadratisch viele Distanzberechnungen notwendig sind. Schon bei zwei euklidischen Dimensionen ist die Berechnung der Distanzmatrix für mehr als die Hälfte der Gesamtlaufzeit verantwortlich. Bei aufwändig zu berechnenden Distanzmaßen wird die Gesamtlaufzeit deshalb stark beeinflusst. Bereits bei fünfzig Dimensionen macht die Laufzeit des NNChain-Algorithmus nur noch etwa ein Zehntel der Gesamtlaufzeit aus (vgl. Abbildung 3.2). In der aktuellen Version unterstützt der Scaffold Hunter zudem das Clustern von Bit-Fingerprints in Kombination mit dem Tanimoto-Distanzmaß. Die Länge eines üblichen Fingerprints beträgt 50 bis einige 1000 Bits. Für die Berechnung der Distanz (Tanimoto-Distanzmaß) zwischen solchen Fingerprints, wird in der Praxis eine deutlich höhere Laufzeit benötigt, als für euklidische Distanzen mit kleiner Dimensionalität. In Zukunft ist für den Scaffold Hunter zudem ein Distanzmaß auf Basis des MCS (Maximum Common Subgraph) der Moleküle geplant. Für dieses NP-schwierige Problem wird eine deutlich höhere Laufzeit als für alle bisher implementierten Distanzmaße erwartet. Die quadratisch vielen Distanzberechnungen sind daher ein kritischer Faktor der Gesamtlaufzeit.

Wie in Abschnitt 2.1.1.5 erwähnt, ist die quadratische Laufzeit für das SAHN Clusterverfahren (für allgemeine Distanzmaße) und Linkage-Verfahren bereits asymptotisch optimal. Die Suche nach einem besseren allgemeinen SAHN-Algorithmus würde die Laufzeit also höchstens um konstante Faktoren verbessern. Zudem ist der NNChain-Algorithmus in der

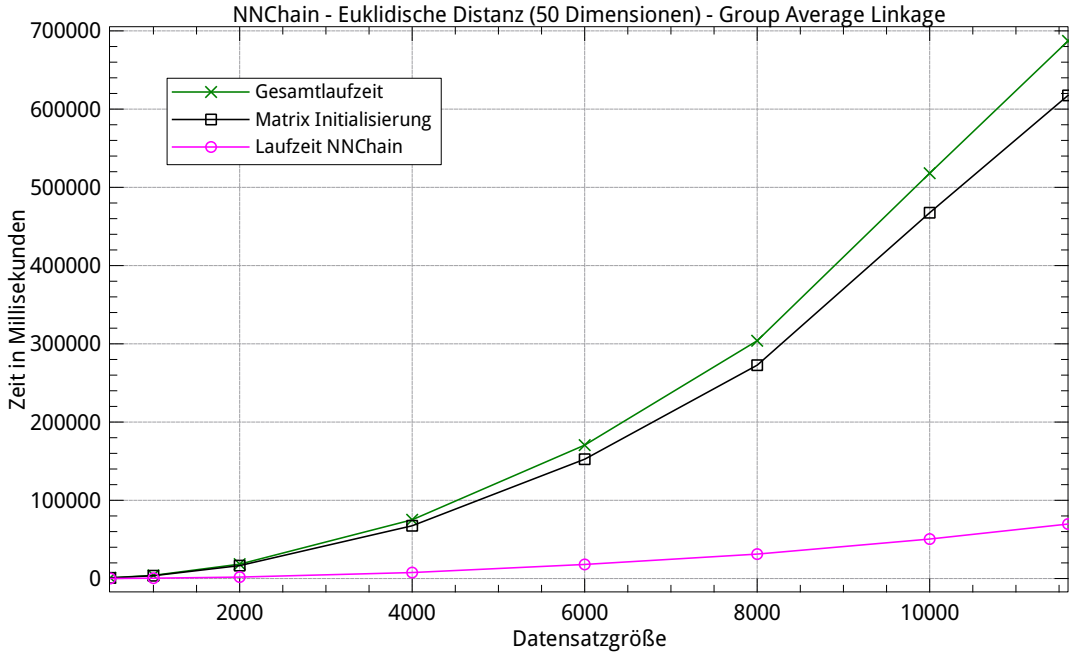



**Abbildung 3.2.:** Laufzeit NNChain, Euklidische Distanz, 50 Dimensionen, Average-Linkage

Praxis schon einer der schnellsten bekannten Algorithmen für allgemeine Distanzmaße (vgl. [Mü11]).

Das Gleiche gilt für die Anzahl der Distanzberechnungen. In der aktuellen Implementierung wird bereits die Symmetrieeigenschaft für Distanzmaße gefordert, so dass nur die Hälfte der Matrix ($\frac{n^2}{2}$ Distanzen) berechnet werden muss. Da auf jede Distanz mindestens einmal zugegriffen werden muss, um den exakten nächsten Nachbar für alle Elemente zu finden, kann die Anzahl der Distanzberechnungen nicht weiter reduziert werden und die Laufzeit für die Matrixinitialisierung ist somit bereits optimal. Da die Matrixinitialisierung für den Großteil der Gesamtlaufzeit verantwortlich ist, wäre die Optimierung der Laufzeit eines Clusterverfahrens, welches auf die Distanzmatrix angewiesen ist, kein Ansatz, der zu einer wesentlichen Reduktion der Laufzeit führt.

## 3.2. Lösungsansätze

Als Lösung des Problems bleibt die Möglichkeit, Indexstrukturen zu verwenden, um die Nächste-Nachbar-Suche zu beschleunigen oder die Problemstellung zu modifizieren und kein (exaktes) SAHN-Clustering zu berechnen. Im zweiten Fall kommt die Verwendung von Heuristiken, das Sampling der Daten oder die Einschränkung auf Spezialfälle in Frage. Um die Kriterien zu definieren, nach denen ein solches Verfahren ausgewählt werden soll, muss

noch einmal auf den Verwendungszweck eingegangen werden. Mögliche Lösungsansätze müssen auf Einschränkungen in Bezug auf die Benutzbarkeit untersucht werden.

Das Clustering des Scaffold Hunter wird vom Benutzer verwendet, um die komplette Ähnlichkeitsstruktur in den Daten zu erkennen. Die Anzahl der Cluster, die gefunden werden sollen ist nicht bekannt. Deshalb sollte auch das beschleunigte Clusterverfahren nach Möglichkeit hierarchisch sein. In Bezug auf die Akzeptanz neuer Verfahren durch den Benutzer ist es sinnvoll, bekannte Verfahren zu verwenden. Dieser Aspekt spräche ebenfalls dafür das bisher verwendete SAHN-Clusterverfahren beizubehalten. In diesem Fall kann die Dendrogrammansicht wiederverwendet werden und der Benutzer muss sich in kein neues Bedienkonzept einarbeiten.

Für den Vektorraum (mit niedriger Dimensionalität) bzw. das euklidische Distanzmaß sind Indexstrukturen zum Beschleunigen der nächsten-Nachbar-Suche bekannt. Bit-Fingerprints in Kombination mit dem Tanimoto-Distanzmaß zählen in der Molekularbiologie jedoch zu der am häufigsten genutzten Datenbasis für das Clustering (s. a. [DB03]). Molekülstrukturen können verwendet werden, um eine Vielzahl von Fingerprints automatisch berechnen zu lassen. Fingerprints stellen deshalb eine einfache Möglichkeit dar, die Moleküle auf Grundlage ihrer Struktur zu vergleichen. Daher hätte eine Einschränkung auf das euklidische Distanzmaß bzw. den Vektorraum eine wesentlich geringere Flexibilität des Clusterverfahrens zur Folge. Graphbasierte Distanzmaße (z. B. mittels Berechnung des MCS) könnten ebenfalls nicht unterstützt werden.

Die Einschränkung auf metrische Distanzmaße ist eine Möglichkeit, die zu keiner Verminderung des aktuellen Funktionsumfanges des Scaffold Hunter führt. Zum einen sind alle bisher implementierten Distanzmaße bereits metrisch. Zum anderen ist die Metrik eine relativ natürliche Einschränkung (s. a. Abschnitt 2.1.2.2), die häufig auch als Qualitätskriterium für Distanzmaße verwendet wird. Deshalb erfüllen die meisten gebräuchlichen Distanzmaße diese Einschränkung.

Heuristische Ansätze verzerren das Clustering und weichen von dem exakten Clustering ab. Bis zu einem gewissen Grad ist diese Verzerrung hinnehmbar, da das Clustering auch im exakten Fall nur eine grobe Klassifizierung der Daten darstellt. Die Datenbasis (Fingerprints, numerische Messwerte, etc.) stellt nur eine Modellierung der Realität dar und bildet daher nur einige Aspekte der Wirklichkeit ab. Zum Beispiel erfassen Fingerprints nur eine fest definierte Menge von Eigenschaften eines Moleküls und stellen eine vereinfachende Sicht auf die Daten dar. Messwerte sind zudem im Allgemeinen fehlerbehaftet. Eine Verzerrung der Realität ist also immer gegeben. Die entscheidenden Fragen lauten daher, wie stark sich diese Verzerrung auf die Aussagekraft des Clustering auswirkt und ob diese Verzerrung zu Fehlentscheidungen des Benutzers führen kann.

Beim Sampling wird nur eine Auswahl $S \subset \mathcal{X}$ (das sogenannte Sample) der Objekte geclustert und dadurch die Eingabegröße reduziert. Das Sampling der Daten hat genau wie bei heuristischen Methoden eine Verzerrung des Clusterings zur Folge. Die Darstellung des

Ergebnisses fordert zudem weitere Anpassungen, da alle Objekte $Y = \mathcal{X} \setminus S$, die nicht Teil des Samples sind, in den Clusterergebnissen fehlen. Eine möglichen Lösung dieses Problems stellt die Zuordnung aller Objekte $y \in Y$ zu einem Sample-Objekt $s \in S$ dar. Anschließend werden diese sogenannten Mikrocluster als Blattknoten des Dendrogramms dargestellt. Die Ähnlichkeitsstruktur der Daten innerhalb dieser Mikrocluster wird nicht berechnet. Somit fehlt die Feinstruktur des Dendrogramms in den SAHN-Clusterergebnissen. Die Mikrocluster können jedoch im Bedarfsfall erneut geclustert und das Ergebnis in das Dendrogramm eingehängt werden.

Ein weiterer wichtiger Aspekt ist die Dimensionalität der Daten. Viele Indexstrukturen funktionieren gut im niedrigdimensionalen Raum und verlieren ihre Laufzeitvorteile bei hoher Dimensionalität (vgl. Abschnitt 4.2.2). Um die Gesamtheit eines Moleküls erfassen zu können, ist es dennoch sinnvoll, die mögliche Dimensionalität des Raumes und die damit mögliche Komplexität des Distanzmaßes nicht zu weit einzuschränken. Bit-Fingerprints in Kombination mit dem Tanimoto-Distanzmaß können z. B. eine intrinsische Dimensionalität (vgl. Gleichung 4.16) aufweisen, die deutlich oberhalb der Effizienzgrenze dieser Indexstrukturen liegt.

## 3.3. Anforderungen an ein alternatives Clusterverfahren

Zusammenfassend lassen sich die folgenden Kriterien für die Wahl der Beschleunigungsmethode festhalten. Die beschleunigte Variante des Clusterverfahrens sollte möglichst viele dieser Kriterien erfüllen:

- Anwendbar auf allgemeine metrische Distanzmaße. Insbesondere keine Beschränkung auf den Vektorraum oder das euklidische Distanzmaß.
- Anwendbar bei aufwändig zu berechnenden Distanzmaßen.
- Anwendbar bei mittlerer bis hoher Dimensionalität der Daten.
- Verwendung bekannter Cluster- und Linkage-Verfahren, um den Benutzern des Scaffold Hunter die Anwendung ohne zusätzliche Einarbeitung zu ermöglichen und die Akzeptanz zu erhöhen.

# Kapitel 4.

# Bekannte Verfahren

Dieses Kapitel gibt einen Überblick über relevante hierarchische Clusterverfahren und Methoden zu deren Beschleunigung.

## 4.1. Hierarchische Clusterverfahren

Es gibt eine Vielzahl unterschiedlicher hierarchischer Clusterverfahren. Dieser Abschnitt gibt deshalb einen Überblick über verschiedene Klassen hierarchischer Clusterverfahren und geht auf die relevanten Algorithmen (in Bezug auf die Anforderungen aus Kapitel 3) näher ein.

### 4.1.1. Dichtebasierte Clusterverfahren

Dichtebasierte Clusterverfahren teilen die Daten in dichte und spärliche Regionen auf. Dichte Regionen beinhalten relativ zu ihrer Ausdehnung eine große Anzahl von Objekten. Für allgemeine metrische Distanzmaße, wird üblicherweise die Anzahl der Nachbarn eines Objektes in einem festgelegten Radius $r$ als Maß für die Dichte verwendet. Die dichten Regionen, welche durch spärliche Regionen voneinander getrennt werden, bilden die Cluster.

Die entstehenden Cluster haben im Gegensatz zu vielen anderen Verfahren eine beliebige Form, die von Cluster zu Cluster variieren kann. Deshalb eignet sich diese Klasse von Clusterverfahren besonders gut, wenn Cluster von nicht einheitlicher Form gefunden werden sollen (s. a. [MC12]). Dichtebasierte Clusterverfahren haben auf Grund der Definition der Cluster in vielen Fällen ebenfalls keine Probleme mit verrauschten Daten, denn nicht alle Objekte müssen zwingend einem Cluster zugeordnet werden. Objekte in den spärlichen Regionen können als Rauschen im Datensatz interpretiert werden.

Beispiele für hierarchische, dichtebasierte Clusterverfahren sind OPTICS [ABK+99], BRIDGE [DLX01], DBCLASD [XEK+98] und DENCLUE [HHK98]. Im Folgenden (Abschnitt 4.1.1.1 und Abschnitt 4.1.1.2) werden DBSCAN [EKS+96] und OPTICS näher vorgestellt. DBSCAN stellt kein hierarchisches Verfahren dar. Da OPTICS jedoch auf diesem Verfahren aufbaut, wird es trotzdem in diesem Kapitel vorgestellt.

#### 4.1.1.1. DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [EKS+96] gehört zu den bekanntesten Vertretern der Klasse dichtebasierter Clusterverfahren. Es basiert auf der Idee von Kernobjekten, Dichteverbundenheit und Dichteerreichbarkeit. Das Verfahren ist mit einem Radius $\epsilon$ und einer Anzahl $MinPts$ parametrisiert.

**4.1.1 Definition (Kernobjekt).** Als Kernobjekte werden Objekte bezeichnet, für die mindestens $MinPts$ Objekte in der $\epsilon$-Nachbarschaft $N_\epsilon$ existieren.

$$N_\epsilon(x_i) = \{x_j \in \mathcal{X} \mid d(x_i, x_j) < \epsilon\}. \tag{4.1}$$

**4.1.2 Definition (Dichteerreichbarkeit).** Ein Objekt $x_j$ ist **direkt** dichteerreichbar von $x_i$ $(i \neq j)$, wenn die folgenden zwei Bedingungen gelten:

- $x_i$ ist Kernobjekt
- $x_j \in N_\epsilon(x_i)$

Ein Objekt $x_j$ ist dichteerreichbar von $x_i$ $(i \neq j)$, wenn es eine Kette von Punkten $x_i, \cdots, x_j$ gibt und jeder Punkt $x_k$ direkt dichteerreichbar von $x_{k-1}$ $(i < k \leq j)$ ist.

**4.1.3 Definition (Dichteverbundenheit).** Zwei Objekte sind dichteverbunden, wenn sie von einem gemeinsamen Objekt aus dichteerreichbar sind (Beispiel: Abbildung 4.1).

Cluster sind so definiert, dass zwei Objekte genau dann in einem Cluster liegen, wenn sie dichteverbunden sind. DBSCAN startet mit einem beliebigen Kernobjekt und erweitert von diesem ausgehend ein Cluster durch ein beliebiges dichteerreichbares Objekt. Ein Cluster ist maximal und zusammenhängend:

**4.1.4 Definition (Maximalität).**

$$x \in C \wedge y \in DichteErreichbar(x) \Rightarrow y \in C \tag{4.2}$$

**4.1.5 Definition (Zusammenhängend).** Für alle Objektpaare in einem Cluster gilt, dass sie dichteverbunden sind.

Als Besonderheit von DBSCAN gilt, dass es Objekte gibt, die weder ein Kernobjekt noch dichteerreichbar sind und damit keinem Cluster angehören. Diese Objekte werden als Noise (Rauschen) bezeichnet und als Fehler in der Datengenerierung interpretiert.

DBSCAN besitzt eine rechnerische Komplexität von $\mathcal{O}(n^2)$ für allgemeine Distanzmaße und benötigt eine Distanzmatrix als Eingabe. In vielen Fällen wird eine Laufzeit von $\mathcal{O}(n\ \log(n))$ genannt. Diese Laufzeit basiert jedoch auf der Annahme einer Indexstruktur, die eine nächste Nachbaranfrage in $\mathcal{O}(\log(n))$ bearbeiten kann. Solche Indexstrukturen sind aber nur für wenige Distanzmaße bekannt. Zudem kann die logarithmische Laufzeit nur bei geringer Dimensionalität der Daten garantiert werden (s. a. Abschnitt 4.2.2).

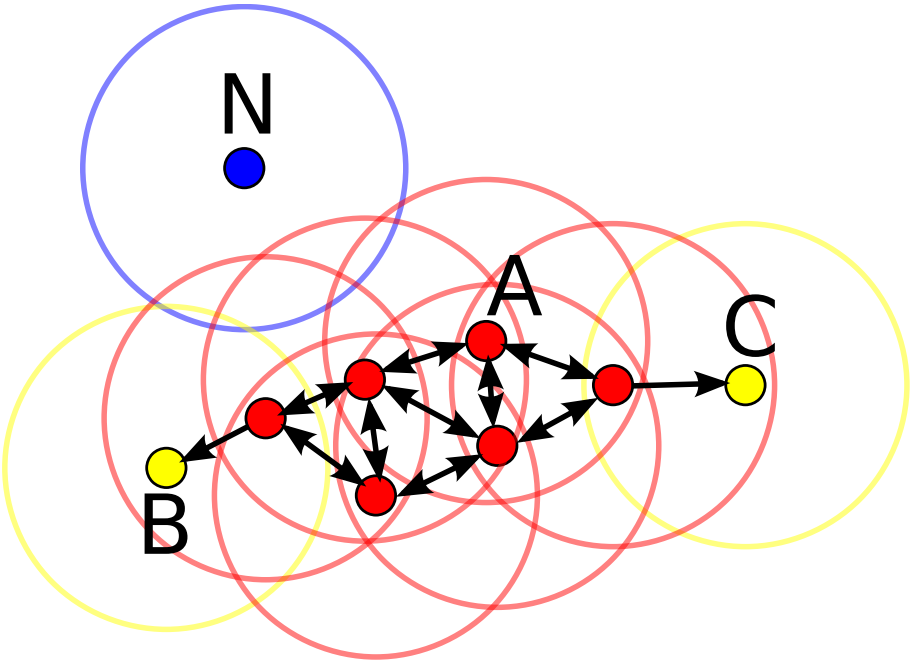



**Abbildung 4.1.:** Dichteerreichbarkeitsgraph: Kern-Objekte (Rot); Dichteerreichbar (Gelb) von A jedoch nicht Kernobjekt; Noise (Blau); $MinPts = 3$; Quelle: Wikipedia

#### 4.1.1.2. OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure) stellt eine hierarchische Erweiterung von DBSCAN dar. In einem Reachability-Plot kann die Dichte der Cluster im Nachhinein variiert werden und dies ermöglicht das einfache Finden der gewünschten Clustergranularität.

Der wesentliche, technische Unterschied besteht in der Reihenfolge, in der Cluster um weitere Objekte erweitert werden. Bei DBSCAN wird mit einem beliebigen Kernobjekt gestartet und von diesem ausgehend ein Cluster durch ein beliebiges dichteerreichbares Objekt erweitert. Im Gegensatz dazu setzt OPTICS einen Min-Heap ein, um die Cluster unter Berücksichtigung einer sogenannten Erreichbarkeits-Distanz zu erweitern. Diese Distanz wiederum basiert auf der Definition der Kern-Distanz (s. a. [ABK+99]). $MinPtsNN(x)$ bezeichnet im Folgenden den Nachbar mit der $MinPts$-kleinsten Distanz ausgehend von Objekt $x$.

**4.1.6 Definition (Kerndistanz, Erreichbarkeitsdistanz).**

$$KernDistanz(x) = \begin{cases} d(x,MinPtsNN(x)) & \text{x ist Kernobjekt} \\ undefiniert & \text{sonst} \end{cases} \tag{4.3}$$

$$ErreichbarkeitsDistanz(x,y) = \begin{cases} max(KernDistanz(y), d(x,y)) & \text{y ist Kernobjekt} \\ undefiniert & \text{sonst} \end{cases} \tag{4.4}$$

Durch die Sortierung nach der Erreichbarkeitsdistanz werden dichte Regionen mit hoher Priorität traversiert. Das führt zu der kompletten Erweiterung eines Clusters bevor die spärlichen Objekte der Umgebung betrachtet werden. Als Ausgabe von OPTICS erhält man einen Erreichbarkeitsplot. Dieser ist ein Balkendiagramm, welches auf der x-Achse die Objekte nach der Reihenfolge ihrer Traversierung durch den Algorithmus aufgeführt hat. Die Höhe der Balken entspricht der Erreichbarkeits-Distanz. Abbildung 4.2 zeigt einen solchen Plot und eine Verlinkung zu einem zweidimensionalen euklidischen Datensatz.

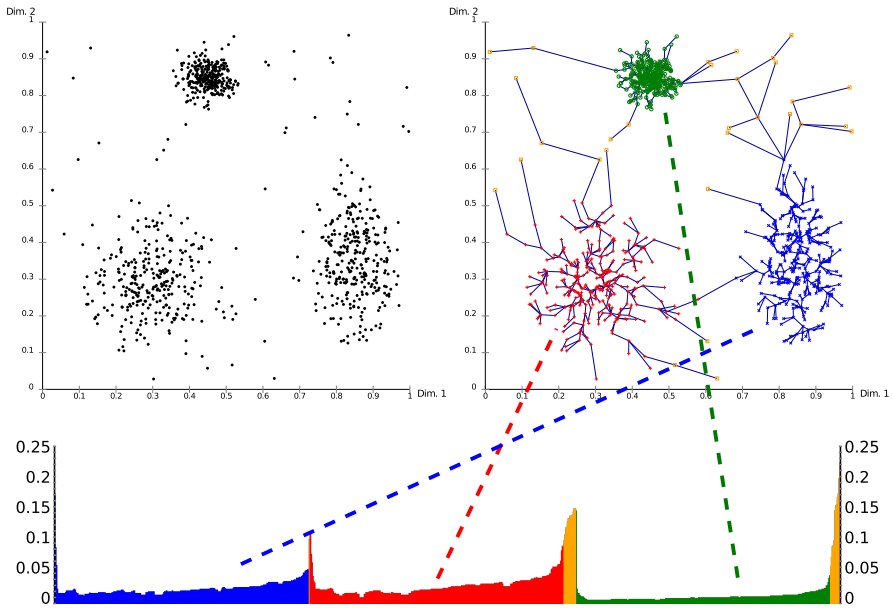



**Abbildung 4.2.:** Erreichbarkeitsplot mit Verlinkung der Daten; Quelle: Wikipedia

Die Täler im Erreichbarkeitsplot sind daher als Cluster zu verstehen. Wenn man eine horizontale Schnittebene durch den Plot zieht, so werden einige Peaks abgeschnitten. Diese Objekte sind als Rauschen zu klassifizieren. Alle Objekte zwischen diesen Peaks sind Elemente eines gemeinsamen Clusters. Dadurch kann die minimale Dichte der Cluster nach der Berechnung – ähnlich wie im Dendrogramm – festgelegt werden.

Eine weitere wichtige Eigenschaft von OPTICS ist, dass nur die Distanzen zu den $MinPts$ nächsten Nachbarn benötigt werden. Für allgemeine Distanzmaße muss für die

Nächste-Nachbar-Suche jedoch trotzdem jede Distanz berechnet werden. In einigen Fällen, kann sie jedoch mit Hilfe von Indexstrukturen (vgl. Abschnitt 4.2.2) oder Eigenschaften des Distanzmaßes (vgl. Abschnitt 4.2.3) in sublinearer Zeit bearbeitet werden. Bei gegebenen nächsten Nachbarn kann die Anzahl der Distanzberechnungen daher auf $n \cdot MinPts$ reduziert werden.

### 4.1.2. Gitterbasierte Clusterverfahren

Gitterbasierte Clusterverfahren erinnern ein wenig an die Idee des *Bucketsort*. Sie unterteilen den Vektorraum in unterschiedliche Zellen und ordnen die Objekte diesen Zellen zu. Später wird meist eine Sortierung der Zellen nach ihrer Dichte vorgenommen und so Zentroiden von Clustern identifiziert. Anschließend werden benachbarte Zellen traversiert, um die Ausdehnung der Cluster festzustellen (s. a. [MC12]).

Zwei bekannte Verfahren, die nach diesem Schema vorgehen, sind STING [WYM97] und GRIDCLUS [Sch96]. Andere Verfahren, wie z. B. OptiGrid [HK99], teilen die Daten diversiv mit Hilfe von Hyperebenen auf. Ein etwas exotisches Verfahren stellt WaveCluster [SCZ00] dar. Es projiziert den beliebig hoch dimensionalen euklidischen Raum in den zweidimensionalen euklidischen Raum, zerteilt diesen in Zellen und interpretiert die Anzahl der Objekte pro Zelle als Graustufe der jeweiligen Zelle. So entsteht ein Graustufenbild, das mit Hilfe der Wavelet-Transformation auf Cluster untersucht wird.

Alle gitterbasierten Clusterverfahren besitzen die gemeinsame nachteilige Eigenschaft einen Vektorraum vorauszusetzen und damit keine freie Wahl des Distanzmaßes zu erlauben. Deshalb kommt diese Klasse von Clusterverfahren nicht als Lösung der Problemstellung in Betracht.

### 4.1.3. SAHN

SAHN-Clusterverfahren sind bereits in Abschnitt 2.1.1.5 vorgestellt worden. Deshalb wird in diesem Kapitel nur auf konkrete Algorithmen eingegangen.

#### 4.1.3.1. NNChain

Der NNChain-Algorithmus bietet die Grundlage der Dendrogrammansicht im aktuellen Scaffold Hunter (vgl. Kapitel 3). Er hat sich in der Praxis als einer der schnellsten verfügbaren exakten SAHN-Clustering-Algorithmen herausgestellt [Mü11]. Eine ausführliche Beschreibung des Algorithmus befindet sich in [Mur85].

Der NNChain-Algorithmus baut auf der Idee auf, dass für bestimmte Linkage-Verfahren und symmetrische Distanzmaße reziproke nächste Nachbarn zu jedem Zeitpunkt verschmolzen werden können, selbst wenn sie, im Gegensatz zu Algorithmus 2.1, nicht die minimale Distanz besitzen. Für ein reziprokes Nächstes-Nachbar-Paar $(x_i, x_j)$ gilt, dass sowohl $x_i$

nächster Nachbar von $x_j$ als auch $x_j$ nächster Nachbar von $x_i$ ist. Damit das Ergebnis korrekt ist, muss das verwendete Linkage-Verfahren die *Reducibility-Eigenschaft* erfüllen.

**4.1.7 Definition (Reducibility-Eigenschaft).** Die Reducibility-Eigenschaft ist erfüllt, wenn für jede Verschmelzung von zwei Clustern $C_i$ und $C_j$ die folgende Implikation erfüllt ist ($C_k$ ist ein beliebiger anderer Cluster):

$$\begin{aligned} & D(C_i, C_j) \leq \inf\{D(C_i, C_k), D(C_j, C_k)\} \\ \Rightarrow & \inf\{D(C_i, C_k), D(C_j, C_k)\} \leq D(C_{i\cup j}, C_k) \end{aligned} \tag{4.5}$$

Alternativ formuliert bewirkt die Reducibility-Eigenschaft, dass der Abstand eines beliebigen Clusters $C_k$, zu den Elementen eines reziproken nächsten-Nachbar-Paares $(C_i, C_j)$, kleiner ist als der Abstand zwischen $C_k$ und dem Cluster $C_{i\cup j}$, welcher aus der Verschmelzung von $C_i$ und $C_j$ hervorgegangen ist.

Die Reducibility-Eigenschaft hat zur Konsequenz, dass die Distanzen mit jedem Verschmelzungsvorgang monoton ansteigen. Es ist garantiert, dass der nächste Nachbar $C_x$ eines Clusters $C_y$ nächster Nachbar bleibt, wenn zwei andere Cluster $C_i$ und $C_j$ ($i$,$j \notin \{x, y\}$) verschmolzen werden. Bei symmetrischen Distanzen gilt zudem, dass die minimale Distanz ein reziprokes nächstes-Nachbar-Paar induziert. Reziproke nächste Nachbarn bleiben deshalb bei erfüllter Reducibility-Eigenschaft während des gesamten Ablaufs reziproke nächste Nachbarn, bis sie selber verschmolzen werden. Damit dürfen sie zu jedem Zeitpunkt verschmolzen werden.

Nicht-monoton ansteigende Distanzen haben in der Dendrogrammdarstellung sogenannte *Reversals* (Umkehrungen) zur Folge. Reversals äußern sich darin, dass ein Dendrogrammknoten niedriger gezeichnet wird als mindestens einer seiner Kinderknoten (vgl. Abbildung 4.3). Bei Linkage-Verfahren, welche die Reducibility-Eigenschaft erfüllen, können Reversals daher nicht auftreten.

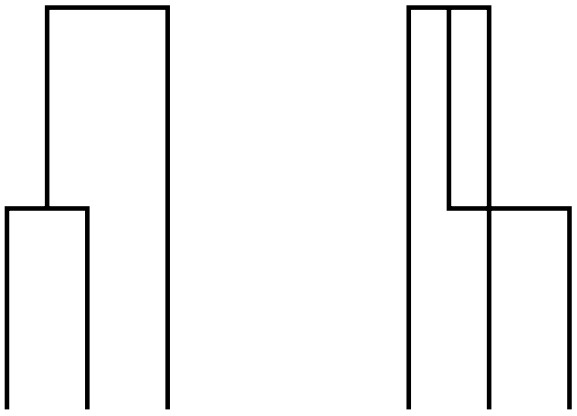


**Abbildung 4.3.:** Dendrogramme ohne und mit Reversal

Alle in Tabelle 2.1 aufgeführten Linkage-Verfahren mit Ausnahme von Median- und Zentroid-Linkage unterstützen die Reducibility-Eigenschaft.

Der Name des NNChain-Algorithmus leitet sich aus der Funktionsweise ab. Der Algorithmus startet mit einem beliebigen Objekt und berechnet den nächsten Nachbarn $C_n$. Anschließend wird der nächste Nachbar von $C_n$ berechnet, usw.. Dadurch wird eine Kette (engl. chain) von nächsten Nachbarn gebildet. Diese wird so lange erweitert bis die letzten beiden Elemente der Nächste-Nachbar-Kette reziprok sind (vgl. Abbildung 4.4). In diesem Fall werden die beiden reziproken Cluster direkt verschmolzen. Anschließend werden die reziproken Cluster aus der Nächste-Nachbar-Kette entfernt und die Berechnung der nächsten Nachbarn für das letzte Kettenglied fortgeführt, so dass die Kette wieder verlängert oder erneut ein reziprokes Nächste-Nachbar-Paar gefunden wird. Dieses Vorgehen wird so lange wiederholt, bis nur noch ein Cluster mit allen Objekten existiert. Sollten die ersten beiden Elemente der Kette reziprok sein und noch mehr als ein Cluster nach deren Verschmelzung vorhanden sein, dann wird eine neue Kette gestartet.

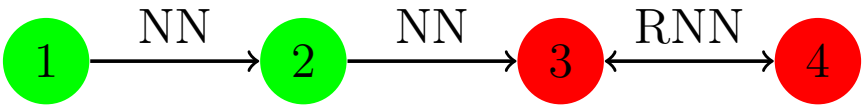



**Abbildung 4.4.:** Nächste-Nachbar-Kette

Die Gesamtlaufzeit des NNChain-Algorithmus ist $\mathcal{O}(n^2)$. Sie teilt sich auf in die Berechnung der Distanzmatrix und $\mathcal{O}(n)$ Nächste-Nachbar-Suchen. Die Begrenzung der Nächste-Nachbar-Suchen durch $\mathcal{O}(n)$ kann folgendermaßen erklärt werden:

Insgesamt ist die Anzahl der Cluster durch $\mathcal{O}(n)$ begrenzt, da das Dendrogramm einen binären Baum mit $n$ Blättern darstellt. Die Nächste-Nachbar-Kette wird genau einmal um jeden Cluster erweitert, da in der Kette kein Cluster zweimal vorkommen kann. Sollte ein Element der Nächste-Nachbar-Kette zweimal gefunden werden, so würde dies ein reziproker nächster Nachbar sein und damit verschmolzen werden. Damit ist die Anzahl der Nächste-Nachbar-Suchen durch $\mathcal{O}(n)$ beschränkt.

Der Speicherbedarf des NNChain-Algorithmus wird durch die Distanzmatrix ($\mathcal{O}(n^2)$) dominiert. Es gibt für Ward-Linkage eine Variante die ohne Distanzmatrix auskommt und mit Hilfe von Clusterzentroiden die Distanz zwischen den Clustern berechnet. Da die Distanzen auf Grund der Symmetrie des Distanzmaßes doppelt berechnet werden, obwohl das (bei Verwendung einer Distanzmatrix) überflüssig ist, ist diese Variante je nach Distanzmaß bis zu einem Faktor von 2 langsamer. Für diese Variante wird allerdings nur linear viel Speicher benötigt, da für jeden Cluster nur der (konstant große) Zentroid gespeichert werden muss und der Speicherplatz für die Nächste-Nachbar-Kette ebenfalls in $\mathcal{O}(n)$ liegt.

#### 4.1.3.2. Dynamic Closest Pair

In der Veröffentlichung [Epp98] werden zwei Container-Datenstrukturen beschrieben, die zu jeder Zeit das minimale reziproke Nächste-Nachbar-Paar für eine Menge von Objekten vorhalten. Es ist möglich Objekte hinzuzufügen oder zu entfernen. Die Datenstruktur funktioniert für beliebige symmetrische Distanzmaße und kann eingesetzt werden, um den naiven SAHN-Clustering-Algorithmus (Algorithmus 2.1) zu beschleunigen. Das zu verschmelzende, minimale reziproke Nächste-Nachbar-Paar ist zu jeder Zeit bekannt. Bei einer Verschmelzung müssen die verschmolzenen Cluster lediglich aus der Datenstruktur entfernt werden und der neu geformte Cluster eingefügt werden.

Die erste vorgestellte Datenstruktur (*FastPair*) kommt mit linear viel Speicher aus und benötigt amortisiert $\mathcal{O}(n \log(n))$ Rechenoperationen pro Einfügevorgang und amortisiert $\mathcal{O}(n \log^2(n))$ Rechenoperationen pro Löschvorgang. Die zweite Datenstruktur benötigt $\mathcal{O}(n^2)$ Speicher und $\mathcal{O}(n)$ Rechenoperationen für einen Einfüge- oder Löschvorgang. Mit Hilfe der zweiten Datenstruktur kann Algorithmus 2.1 leicht modifiziert werden, um die gleiche theoretische Laufzeit wie NNChain zu erreichen. Zudem werden alle Linkage-Verfahren unterstützt.

FastPair ist eine Abwandlung der *Conga-Line*-Datenstruktur aus [Epp95]. Sehr stark verkürzt ausgedrückt wird für jedes Objekt der nächste Nachbar vorgehalten. Durch das Scannen aller dieser Nächste-Nachbar-Beziehungen kann in linearer Zeit die kleinste Distanz gefunden werden und aufgrund der Symmetrie ist dieses Objektpaar reziprok (vgl. Abschnitt 4.1.3.1). Durch den Aufbau der Datenstruktur kann die Neuberechnung der Nächste-Nachbar-Beziehungen stark reduziert werden. Von Zeit zu Zeit wird die komplette Datenstruktur neu aufgebaut. Die Datenstruktur benötigt amortisiert $\mathcal{O}(n \log(n))$ Rechenoperationen pro Einfügevorgang und amortisiert $\mathcal{O}(n \log^2(n))$ Rechenoperationen pro Löschvorgang. Obwohl sie mit linearem Speicherbedarf auskommt, ist sie für unseren Verwendungszweck uninteressant, da für die meisten Linkage-Verfahren die quadratische Distanzmatrix benötigt wird und daher der Speicherbedarf nur für einige Spezialfälle effektiv linear ist. Für alle Linkage-Verfahren, die mit Hilfe eines Repräsentanten die Distanzen berechnen können, kann der Speicherbedarf des naiven Clustering-Algorithmus auf $\mathcal{O}(n)$ gesenkt werden. Es gilt jedoch genau wie bei der Variante des NNChain-Algorithmus mit linearem Speichergebrauch (s. a. Abschnitt 4.1.3.1), dass die Distanzen mehrfach berechnet werden müssen. Im Gegensatz zu NNChain muss die Distanz aber deutlich häufiger (nicht nur doppelt) berechnet werden, da die zugrunde liegende Datenstruktur mehrfach neu aufgebaut wird. Damit sollte die Gesamtlaufzeit bei rechenintensiven Distanzmaßen und ohne die Verwendung der Distanzmatrix schlechter sein (vgl. Argumentation aus Abschnitt 3.1). Im Gegensatz zu NNChain unterstützt diese Datenstruktur auch Median- und Zentroid-Linkage. Damit wäre sie höchstens für diese Linkage-Verfahren im seltenen Fall von stark begrenztem Speicher und sehr hoher Rechenleistung empfehlenswert.

Die zweite Datenstruktur ist eine Art *Quadtree* über der Distanzmatrix. Jeder Knoten des Quadtree referenziert die minimale Distanz des Unterbaumes bzw. des zugehörigen Teils der Distanzmatrix. Um Einfüge- und Löschoperationen zu unterstützen wird diese Baumstruktur mengentheoretisch definiert.
Jede Menge $S(i,j)$ ist eine Teilmenge der Objekte $\mathcal{X} = \{x_0, \ldots, x_{n-1}\}$:

$$S(i,j) = x_{i2j}, x_{i2j+1}, \ldots, x_{(i+1)2j-1} \tag{4.6}$$

$S(i,0)$ entspricht den Singleton Clustern $C_i = \{x_i\}$ und $S(i,j)$ entspricht der disjunkten Vereinigung aus $S(2i, j-1)$ und $S(2i+1, j-1)$. Bei einer Einfügeoperation wird das neue Element mit $x_n$ bezeichnet. Durch das Löschen eines Objektes $x_i$ wird das letzte Objekt $x_{n-1}$ umbenannt in $x_i$.
Die Knoten des Baumes werden als $D(i,j,k)$ modelliert und entsprechen der minimalen Distanz aller Tupel von Objekten aus $S(i,j) \times S(i,k)$. Die Anzahl der eindeutigen Mengen $S(i,j)$ ist durch $2n-1$ beschränkt und die Menge aller Knoten $D(i,j,k)$ durch $\frac{2}{3}n^2$. Jede Einfüge- oder Löschoperationen verändert $\mathcal{O}(n)$ Knotenwerte. Die Werte können in konstanter Zeit inkrementell berechnet werden:

$$\begin{aligned} D(i,j,k) = \min\{&D(2i, j-1, 2k), D(2i+1, j-1, 2k), \\ &D(2i, j-1, 2k+1), D(2i+1, j-1, 2k+1)\} \end{aligned} \tag{4.7}$$

In [Epp98] wurde gezeigt, dass die Laufzeit der Quadtree-Datenstruktur der Laufzeit der FastPair-Datenstruktur in der Praxis überlegen ist. Verbesserungen an der FastPair-Datenstruktur in [CE04] könnten dieses Verhältnis jedoch umkehren. In [CE04] ist aber kein direkter Vergleich mit der Quadtree-Datenstruktur vorgenommen worden.

Leider sind in der Literatur zwischen dieser Datenstruktur und NNChain keine Laufzeitvergleiche zu finden. Ein großer Laufzeitvorteil gegenüber NNChain kann jedoch nicht erwartet werden, da die Initialisierung der Distanzmatrix bereits den Großteil der Gesamtlaufzeit ausmacht und diese bei beiden Algorithmen erforderlich ist.

#### 4.1.3.3. Generic Clustering (Müllner)

Der Generic-Clustering-Algorithmus [Mü11] unterstützt im Gegensatz zum NNChain-Algorithmus alle Linkage-Verfahren. Von dieser Eigenschaft ist auch sein Name abgeleitet. Die Laufzeit ist durch $\mathcal{O}(n^3)$ und $\Omega(n^2)$ beschränkt. Der Speicherplatzbedarf ist auf Grund der Verwendung einer Distanzmatrix quadratisch. Trotz seiner schlechten oberen Schranke für die Laufzeit ist der Algorithmus in der Praxis fast genau so schnell wie der NNChain-Algorithmus und die praktische Laufzeit orientiert sich an der unteren asymptotischen Schranke (s. a. [Mü11]).

Die zentrale Idee des Algorithmus besteht darin die Nächste-Nachbar-Suchen so weit wie möglich aufzuschieben. Im Verlauf des SAHN-Clustering verringert sich die Clusteranzahl mit jeder Verschmelzung. Daher ist die Nächste-Nachbar-Suche zu einem späteren

Zeitpunkt weniger aufwändig. In einigen Fällen ist die Berechnung der nächsten Nachbarn von einem Objekt $x$ gar nicht mehr notwendig, da $x$ vorher – in der Rolle des nächsten Nachbarn eines weiteren Objekts – verschmolzen wurde. Der Algorithmus verwendet eine Prioritätswarteschlange $Q$, in welche alle Objekte sortiert nach ihrer Distanz zum nächsten Nachbar eingefügt werden. Dadurch ist das zu verschmelzende, minimale reziproke Nächste-Nachbar-Paar immer das kleinste Element. Der genaue Ablauf ist in Algorithmus 4.1 festgehalten. Der Algorithmus verwendet die folgenden Datenstrukturen:

**Q:** Prioritätswarteschlange von Objekten (s. o.)

**size:** Array, in dem die Größen der Cluster gespeichert werden

**NN:** Array, in dem die nächsten Nachbarn der Cluster gespeichert werden

**mindist:** Array, in dem die Distanzen zum nächsten Nachbarn gespeichert werden

**d:** Distanzmatrix

Die Nächste-Nachbar-Suche wird in diesem Algorithmus immer nur vorwärts vorgenommen. In einem Array von indexierten Objekten wird der nächste Nachbar daher nur in den Objekten mit größeren Indizes gesucht. Durch die Symmetrie des Distanzmaßes steht das Objekt mit der kleinsten Distanz zu seinem nächste Nachbar trotzdem immer am Anfang von $Q$ (entweder als Objekt für das der nächste Nachbar gesucht wurde oder in der Rolle des nächsten Nachbarn). Dieser Trick beschleunigt die Nächste-Nachbar-Suche um einen konstanten Faktor.

Werden zwei Cluster $C_i$ und $C_j$ (mit $\text{Index}(C_i) < \text{Index}(C_j)$) verschmolzen, so wird $C_j$ einfach durch $C_{i\cup j}$ ersetzt und $C_{i\cup j}$ besitzt daraufhin den gleichen Index wie $C_j$. Damit werden die Nächste-Nachbar-Beziehungen von anderen Clustern $C_k$, für welche $NN(C_k) = C_j$ gilt, automatisch auf $NN(C_k) = C_{i\cup j}$ geändert. Alle Nächste-Nachbar-Beziehungen der Art $NN(C_k) = C_i$ werden in der Schleife in Zeile 28 nach dem gleichen Schema angepasst. Die Distanzen dieser Nächste-Nachbar-Beziehungen werden einfach übernommen. Nur falls diese Distanz nicht mehr stimmt, muss der nächste Nachbar neu berechnet werden. Dies geschieht jedoch erst wenn der entsprechende Cluster aus $Q$ entfernt wird. Insbesondere bei Single- und Complete-Linkage bleibt die Distanz eines beliebigen Clusters $C_k$ zu $C_{i\cup j}$ häufig identisch zu der Distanz $D(C_k, C_j)$. Dieses Vorgehen spart einige Nächste-Nachbar-Suchen.

Ein Problem stellt die Vorwärtssuche in Kombination mit der Ersetzungsstrategie jedoch bei der Verwendung von Linkage-Verfahren dar, welche die Reducibility-Eigenschaft nicht erfüllen. In diesem Fall kann es vorkommen, dass die Distanz $D(C_k, C_{i\cup j})$ kleiner ist als $D(C_k, C_i)$ oder $D(C_k, C_j)$. Falls der Index von $C_k$ zusätzlich kleiner als der von $C_{i\cup j}$ ist, wird durch die Vorwärtssuche die Distanz $D(C_k, C_{i\cup j})$ nicht berücksichtigt. Dadurch würden das Objektpaar $(C_k, C_{i\cup j})$ unter Umständen zu spät verschmolzen, da es in $Q$

noch mit der Distanz $D(C_k, C_i)$ oder $D(C_k, C_j)$ einsortiert wurde. Die Schleife in Zeile 33 korrigiert diesen Fall und führt die Nächste-Nachbar-Suche sofort für entsprechende Objekte durch. Anschließend wird $Q$ aktualisiert.

Die Laufzeit des Algorithmus setzt sich folgendermaßen zusammen: Bei der Initialisierung von $Q$ muss für jedes Objekt ein nächster Nachbar berechnet werden. Wenn für die Nächste-Nachbar-Suche eine Laufzeit von $\mathcal{O}(n)$ angenommen wird, dann beträgt die Gesamtlaufzeit für die Initialisierung $\mathcal{O}(n^2)$. Das Einfügen jedes einzelnen Objekts (der insgesamt $n$ Objekte) in $Q$ ist in logarithmischer Zeit möglich und fällt deshalb, asymptotisch gesehen, nicht ins Gewicht.

1: **procedure** GENERIC_LINKAGE($N, d$) ▷ $N$: Anzahl Objekte, $d$: Distanzen
2: $S \leftarrow (0, \ldots, N-1)$
3: $L \leftarrow [\,]$ ▷ Ausgabeliste
4: $size[x] \leftarrow 1$ for all $x \in S$
5: **for** $x$ in $S \setminus \{N-1\}$ **do** ▷ Generiere Liste der NN
6: $NN[x] \leftarrow \text{argmin}_{y>x}\, d[x, y]$
7: $mindist[x] \leftarrow d[x, NN[x]]$
8: **end for**
9: $Q \leftarrow$ (priority queue of indices in $S \setminus \{N-1\}$, keys are in $mindist$)
10: **for** $i \leftarrow 1, \ldots, N-1$ **do** ▷ Hauptschleife
11: $a \leftarrow$ (minimal element of $Q$)
12: $b \leftarrow NN[a]$
13: $\delta \leftarrow mindist[a]$
14: **while** $\delta \neq d[a, b]$ **do** ▷ Neuberechnung der NN (wenn notwendig)
15: $NN[a] \leftarrow \text{argmin}_{x>a}\, d[a, x]$
16: Update $mindist$ and $Q$ with $(a, d[a,NN[a]])$
17: $a \leftarrow$ (minimal element of $Q$)
18: $b \leftarrow NN[a]$
19: $\delta \leftarrow mindist[a]$
20: **end while**
21: Remove the minimal element $a$ from $Q$.
22: Append $(a, b, \delta)$ to $L$. ▷ Verschmelze minimales reziprokes NN-Paar
23: $size[b] \leftarrow size[a] + size[b]$ ▷ Verwende $b$ als Index für den neuen Cluster
24: $S \leftarrow S \setminus \{a\}$
25: **for** $x$ in $S \setminus \{b\}$ **do** ▷ Update der Distanzmatrix
26: $d[x,b] \leftarrow d[b,x] \leftarrow lwformula(d[a, x], d[b, x], d[a, b], size[a], size[b], size[x])$
27: **end for**

**Algorithmus 4.1:** Generic-Clustering-Algorithmus, Originale Version (Teil 1)

```
28:         for x in S such that x < a do                ▷ Aktualisierung der NN-Kandidaten
29:             if NN[x] = a then                        ▷ Aufschub der NN-Suche
30:                 NN[x] ← b
31:             end if
32:         end for
33:         for x in S such that x < b do
34:             if d[x, b] < mindist[x] then
35:                 NN[x] ← b
36:                 Update mindist and Q with (x, d[x, b])    ▷ Erhalte untere Schranke
37:             end if
38:         end for
39:         NN[b] ← argmin_{x>b} d[b, x]
40:         Update mindist and Q with (b, d[b,NN[b]])
41:     end for
42:     return L                                         ▷ Das Dendrogramm, (((N − 1) × 3)-Matrix)
43: end procedure
```

**Algorithmus 4.1:** Generic-Clustering-Algorithmus, Originale Version (Teil 2)

Die Hauptschleife wird insgesamt $n$ mal durchlaufen, da in jedem Durchlauf zwei Cluster zu einem neuen verschmolzen werden. Die Laufzeit des Algorithmus ist an durch die Schleife in Zeile 14 stark davon abhängig wie die Nächste-Nachbar-Struktur in den Daten beschaffen ist. Im schlechtesten Fall haben alle Objekte, bis auf eines, immer einen gemeinsamen nächsten Nachbar. Damit würden auch alle Einträge aus $Q$ ungültig und es müssten in Zeile 15 unter Umständen für alle Objekte neue nächste Nachbarn berechnet werden. Die Gesamtlaufzeit wäre in diesem Fall kubisch. Im besten Fall würde das erste Element aus $Q$ immer gültig sein. Zum Beispiel könnte der verschmolzene Cluster immer das kleinste Element in $Q$ sein und für diesen wurde in dem vorherigen Schleifendurchlauf immer ein gültiger Nachbar berechnet. Dann wäre die Gesamtlaufzeit quadratisch. Die Laufzeiten der Schleifen in Zeile 28 und 33 sind immer linear, falls *DecreaseKey* in Zeile 40 in $\mathcal{O}(1)$ ausgeführt werden kann, und fallen damit asymptotisch nicht ins Gewicht.

In Abschnitt 5.3.1 werden Anpassungen am Algorithmus vorgenommen, durch welche die Schleifen in den Zeilen 28 und 33 wegfallen. In diesem Fall kann die untere Schranke der Laufzeit des Algorithmus durch eine Nächste-Nachbar-Suche in sublinearer Zeit verbessert werden.

#### 4.1.3.4. Single-Linkage-MST-Algorithmus

Für Single-Linkage existiert ein spezieller Algorithmus [Roh73] der auf der Berechnung des MST (Minimaler Spannbaum) des Distanzgraphen basiert. Das grundlegende Vorgehen des Algorithmus lässt sich in drei Schritten beschreiben:

1. Berechne den MST des (vollständigen) Distanzgraphen.
2. Sortiere die Kanten des MST nach ihrer Distanz.
3. Verschmelze die Cluster $C_i$ und $C_j$ in der Reihenfolge der MST-Kanten $(x_i, x_j)$ mit $x_i \in C_i$ und $x_j \in C_j$.

Die Laufzeit des Algorithmus wird durch Schritt 1 dominiert. Für einen vollständigen Graphen benötigt der Algorithmus von Prim eine Laufzeit von $\mathcal{O}(n^2)$. Schritt 2 kann in $\mathcal{O}(n \ \log(n))$ und Schritt 3, mit Hilfe der Union-Find-Datenstruktur, praktisch in amortisiert linearer Zeit (inversen Ackermannfunktion als Faktor) durchgeführt werden.

Ein Korrektheitsbeweis befindet sich in [Mü11]. Er wird in dieser Arbeit nicht vorgestellt, da der Single-Linkage-MST-Algorithmus nur der Vollständigkeit halber aufgeführt wurde und im weiteren Verlauf der Arbeit keine große Rolle spielt.

## 4.2. Bekannte Ansätze zur Beschleunigung

Dieser Abschnitt beschreibt Ansätze zur Beschleunigung beliebiger Clusterverfahren, welche Teilprobleme optimieren, die vielen Clusterverfahren gemeinsam sind oder die Eingabe in einem Vorverarbeitungsschritt modifizieren. Die Auswahl der vorgestellten Verfahren und der Umfang der Darstellung orientiert sich an der Eignung für die Problemstellung. Zudem wird ein möglichst breites Spektrum an unterschiedlichen Ansätzen abgedeckt.

### 4.2.1. Datenverdichtung

Unter Datenverdichtung wird ein Vorverarbeitungsschritt verstanden, der die Eingabegröße für einen Algorithmus reduziert und damit die Gesamtlaufzeit der Berechnung vermindert. Ein Nachteil, der alle Datenverdichtungsverfahren betrifft, ist die durch die Datenkompression verloren gehende Feinstruktur der Daten. Zudem muss, falls ein solches Verfahren im Scaffold Hunter eingesetzt werden soll, die Darstellung des Dendrogramms angepasst werden, um ganze Mikrocluster als Blätter zu unterstützen. Ein späteres Clustern dieser Mikrocluster könnte im Bedarfsfall einige dieser Nachteile mindern. In diesem Abschnitt werden einige relevante Techniken zur Datenverdichtung vorgestellt.

#### 4.2.1.1. Sampling

Beim Sampling wird die Reduktion der Datengröße durch die Auswahl einer Stichprobe von Objekten $S \subset \mathcal{X}$ erreicht. Welche Objekte im Sample $S$ enthalten sind, entscheidet die

Sampling-Strategie. Das Ziel der Sampling-Strategie ist es, die Grundgesamtheit unverzerrt zu erfassen.

**Zufälliges Sampling** Eine der einfachsten Strategien ist das zufällige (*Random*)-Sampling (s. a. [SYM10]). Beim zufälligen Sampling wird eine festgelegte Anzahl von Objekten aus den Daten mit gleichverteilter Wahrscheinlichkeit gezogen. Sind die Daten stochastisch unabhängig, ist das zufällige Sampling ein unverzerrter Schätzer der zugrundeliegenden Dichtefunktion. Die Varianz nimmt mit zunehmender Größe von $S$ ab.

**Strukturiertes Sampling** Wenn die Daten nicht stochastisch abhängig sind und Informationen über die Verteilung der Daten verfügbar sind, dann führen strukturierte Sampling-Strategien zu besseren Ergebnissen. Durch die Vorkenntnis über die Struktur der Daten, kann z. B. garantiert werden, dass Teilmengen anteilig korrekt in der Stichprobe repräsentiert sind.

**Cluster Sampling** Erwähnenswert ist, dass Clustering selbst eine strukturierte Sampling-Strategie darstellen kann. Dazu wird mit einem möglichst schnellen Clusterverfahren die gesamte Datenmenge geclustert. Anschließend ist es möglich, eine (der Clustergröße entsprechende) Anzahl von Objekten aus jedem Cluster für das Sample zu verwenden und anschließend ein aufwändiges Clustering über dem Sample zu berechnen. Dieses Vorgehen hat den Vorteil, dass alle relevanten Gruppen, also auch kleine, durch das Sample erfasst werden. Diese Strategie wird bereits zur Beschleunigung von Clusterverfahren eingesetzt. Zum Beispiel verwendet der *Leaders-AL*-Algorithmus (s. a. [PHB+10]) ein sehr schnelles, für metrische Distanzmaße optimiertes, Leaders-Clusterverfahren, um ein Sample zu erstellen. Dieses Sample wird dann erneut mit einem SAHN-Clusterverfahren und Average-Linkage geclustert.

#### 4.2.1.2. Data Bubbles

Wenn Datenverdichtungsverfahren wie das Sampling eingesetzt werden, um ein hierarchisches Clusterverfahren zu beschleunigen, ergeben sich strukturelle Verzerrungen. Für die klassischen SAHN-Clusterverfahren lässt sich anhand von Single-Linkage gut zeigen, weshalb eine solche Verzerrung entsteht.

Man kann ein Sample-Objekt als Repräsentant für eine Menge von Objekten der Grundmenge $\mathcal{X}$ auffassen und daher ist es naheliegend, die Objekte aus $\mathcal{X}$ jeweils dem nächsten Sample-Objekt aus $S$ zuzuordnen. Diese Mengen von Objekten, die zu einem Sample-Objekt gehören, werden Mikrocluster genannt. Im Fall von Single-Linkage würde die Distanz zwischen zwei Mikroclustern $M_1$ und $M_2$ immer überschätzt werden, denn die Single-Linkage-Distanz zwischen $M_1$ und $M_2$ ist die minimale Distanz der Objektpaare aus $M_1 \times M_2$. Es ist unwahrscheinlich, dass das Objektpaar mit der minimalen Distanz

den Sample-Objekten entspricht. Zudem wird die Distanz nicht gleichmäßig überschätzt und der Grad der Verzerrung ist abhängig von der Ausdehnung des Mikroclusters in Richtung des anderen Mikroclusters (vgl. Abbildung 4.5). Für Complete-Linkage existiert das gleiche Problem, nur dass die Distanzen unterschätzt werden. OPTICS basiert auf der k-nächsten Nachbarschaft eines Objekts und ist daher ebenfalls von dieser Art der Verzerrung betroffen.

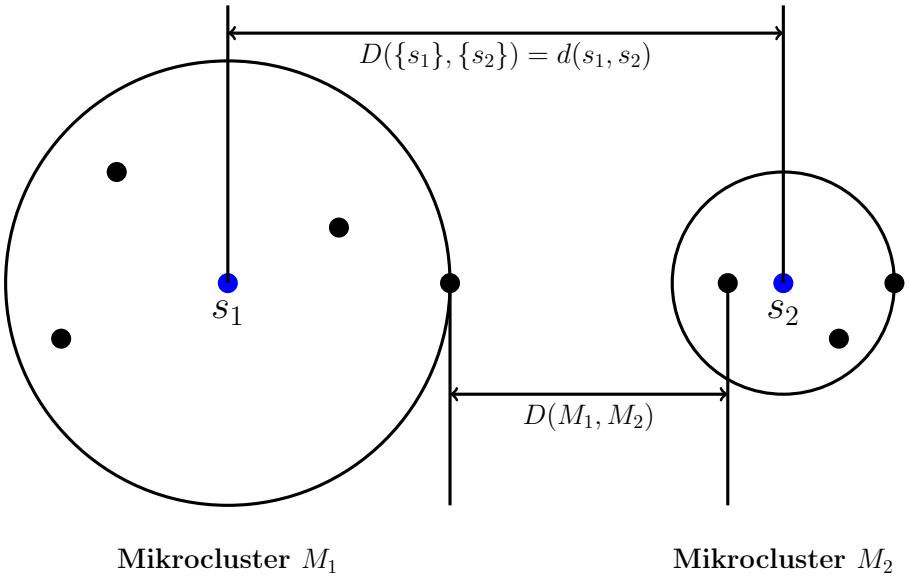



**Abbildung 4.5.:** Mikrocluster Verzerrung bei Single-Linkage

Man könnte diese Verzerrungen leicht auflösen, indem die tatsächliche Distanz zwischen den zwei Mikroclustern berechnet wird. Das Problem dabei sind die quadratisch vielen Distanzberechnungen. Diese würden den Laufzeitvorteil des Sampling zunichte machen. Deswegen verwenden BIRCH [ZRL96], und Data Bubbles [BKK+01; ZS03] Statistiken über die Mikrocluster, mit deren Hilfe die Mikrocluster-Distanz abgeschätzt werden kann. BIRCH und die erste Variante von Data Bubbles [BKK+01] sind auf den Vektorraum eingeschränkt und kommen daher als Lösung der Problemstellung nicht in Frage. In einer zweiten Variante des Data-Bubbles-Ansatzes [ZS03] wird nur die Metrik als Einschränkung des Distanzmaßes gefordert. Das Data-Bubbles-Datenverdichtungsverfahren wird in beiden zugrunde liegenden Veröffentlichungen zusammen mit dem OPTICS-Clusterverfahren (vgl. Abschnitt 4.1.1.2) verwendet. Es lässt sich aber problemlos auch für Single-Linkage-SAHN-Clusterverfahren verwenden.

Um die Idee des Data-Bubbles-Ansatzes für allgemeine metrische Distanzmaße zu erklären wird zunächst die Variante für den Vektorraum vorgestellt. Ein Data Bubble $B_i$ entspricht dem Mikrocluster $M_i$ und besteht aus drei Informationen:

**rep** Als Repräsentant eines Bubbles wird der Zentroid gewählt

**n** Die Kardinalität des Mikroclusters $M_i$

**extent** Radius von $B_i$, der alle Objekte in $M_i$ umfasst

Die Distanz zwischen zwei Data Bubbles $B_i$ und $B_j$ berechnet sich dann folgendermaßen:

$$\begin{aligned} D(B_i, B_j) =& d(rep_{B_i}, rep_{B_j}) - extent_{B_i} - extent_{B_j} \\ &+ avgNnDist(B_i) + avgNnDist(B_j) \end{aligned} \tag{4.8}$$

Dabei ist *avgNnDist* eine Funktion, welche die durchschnittliche nächste-Nachbar-Distanz innerhalb des Mikroclusters berechnet. Unter der Annahme der Gleichverteilung von Objekten innerhalb der Mikrocluster und einem $d$-dimensionalen Vektorraum kann diese Distanz folgendermaßen geschätzt werden:

$$avgNnDist(B_i) = \left(\frac{1}{n}\right)^{\frac{1}{d}} \cdot extent \tag{4.9}$$

Wenn man Abbildung 4.5 genauer betrachtet, ist ersichtlich warum auf diese Art und Weise vorgegangen wird. Würde man von der Distanz zwischen den beiden Repräsentanten nur den *extent* abziehen, würde die tatsächliche Distanz unterschätzt werden, da ein Objekt nicht genau auf dem äußeren Rand der Sphäre in Richtung des anderen Clusters liegen muss. Die durchschnittliche nächste-Nachbar-Distanz ist ein Schätzwert für diese Abweichung.

Im allgemeinen metrischen Raum funktioniert diese Methode nicht. Es existieren hier nur paarweise Distanzen und keine geometrischen Objekte. Zudem kann kein Zentroid berechnet werden und die Berechnung eines Medoid ist sehr rechenintensiv. Aus diesen Gründen wird das ursprüngliche Sample-Objekt $s_i$ als Repräsentant von $B_i$ gewählt.

Da das Sample-Objekt am Rand des Mikroclusters liegen kann und der zugrundeliegende Raum nicht dem Vektorraum entspricht, ist das Konzept einer Sphäre nicht länger ein adäquates Werkzeug zum Schätzen der Ausdehnung des Mikroclusters. Deshalb wird es durch ein Konzept ersetzt, welches die Ausdehnung eines Bubbles $B_i$, ausgehend vom Repräsentanten $s_i$, in die Richtung eines anderen Bubbles $B_j$ bestimmt.

$$BorderDist_{B_i}(B_j) = d(s_i, s_j) - \min_{o \in B_i}(d(o, s_j)) \tag{4.10}$$

Die Distanz zwischen den zwei Bubbles berechnet sich dann analog zu Gleichung 4.8, jedoch ohne den Durchschnittsterm. In [ZS03] wurden noch einige weitere Korrekturen der Distanz, insbesondere in Bezug auf das OPTICS-Clusterverfahren, vorgestellt. Auf diese Korrekturen wird in dieser Arbeit jedoch nicht weiter eingegangen, da sie im weiteren Verlauf nicht benötigt werden und den Umfang dieser Arbeit sprengen würden.

In der Praxis haben sich Data Bubbles für OPTICS mit allgemeinen metrischen Distanzmaßen als bessere Alternative zu zufälligem Sampling erwiesen. Aufgrund der Ähnlichkeit sollte sich dieser Vorteil auch für Single-Linkage-Verfahren bestätigen. Die Laufzeit ist um einen kleinen Faktor ($< 2$) langsamer als bei der Anwendung von zufälligem Sampling. Verwendet man genau so viele Samples für das zufällige Sampling, dass die Laufzeit

gleichauf mit der des Data-Bubbles-Ansatzes ist, so ist die Qualität des Data-Bubbles-Datenverdichtungsverfahrens immer noch deutlich besser als im Falle des Samplings. Interessanterweise gilt diese Beobachtung auch im Vergleich mit der Data-Bubbles-Variante für den Vektorraum.

Ein Nachteil des Data-Bubble-Ansatzes besteht in der auf Single-Linkage beschränkten Einsetzbarkeit. Für Complete-Linkage lässt sich das Verfahren jedoch einfach anpassen, in dem die Ausdehnung eines Bubbles in die entgegengesetzte Richtung verwendet wird:

$$BorderDistCompleteLinkage_{B_i}(B_j) = d(s_i, s_j) - \max_{o \in B_i}(d(o, s_j)) \tag{4.11}$$

### 4.2.2. Indexstrukturen

Indexstrukturen sind ein beliebtes Mittel um bestimmte Anfragen an Datenbanken zu beschleunigen. Sie lassen sich ebenfalls einsetzen, um die Nächste-Nachbar-Suche zu beschleunigen. Die meisten Indexstrukturen beschränken sich jedoch auf den euklidischen Vektorraum.

Ein populäres Beispiel sind R-Bäume (s. a. [Gut84]). Mit ihnen können Bereichsabfragen effizient beantwortet werden. Der Vektorraum wird in rechteckige Bereiche eingeteilt, die über eine B-Baum-artige Struktur verwaltet werden. Leider ist die Effizienz von R-Bäumen stark abhängig von der Dimensionalität der Daten: Bei hoher Dimensionalität überlappen sich die Bereiche des R-Baumes zunehmend. Die in niedrigen Dimensionen erwartete logarithmische Laufzeit für die Nächste-Nachbar-Suche ist bei hohen Dimensionen nicht zu halten. Deshalb wurden verschiedene Varianten des R-Baumes entwickelt. Zum Beispiel vermindern die R*-Bäume [BKS+90] die Überlagerungen der Bereiche und verbessern damit das Verhalten bzgl. der Dimensionalität.

Ein weiteres Beispiel für den Vektorraum ist das Locality-Sensitive Hashing (LSH) [IM98]. LSH ist eine approximative Nächste-Nachbar-Suche und wurde bereits für das Single-Linkage-SAHN-Clustering eingesetzt (s. a. [KHI+07]). Die Laufzeit beträgt $\mathcal{O}(nB)$, wobei $B$ die Anzahl der *Buckets* darstellt. Bei einer festen Anzahl von Buckets ist der Algorithmus sehr schnell und der Laufzeitvorteil macht sich bereits bei kleinen Datensätzen stark bemerkbar. Die Clusteringqualität ist jedoch von der Anzahl der Buckets abhängig und ihre Anzahl muss daher für große Datensätze gesteigert werden, um die gleich Qualität zu erreichen.

Für den allgemeinen metrischen Fall sind die Möglichkeiten noch weiter eingeschränkt. Auf diesem Gebiet sind keine allgemeinen Indexstrukturen bekannt, welche die Nächste-Nachbar-Suche asymptotisch beschleunigen [ZS06]. In [DGS+03] wird eine Indexstruktur für metrische Distanzmaße auf Basis des Pivot-Ansatzes (vgl. Abschnitt 4.2.3) in Kombination mit Hashing vorgeschlagen. Diese scheint zumindest in einigen Fällen die Laufzeit um konstante Vorfaktoren zu beschleunigen.

### 4.2.3. Pivot-Ansatz

Der Pivot-Ansatz verwendet die Dreiecksungleichung und die Symmetrie metrischer Distanzmaße, um Distanzen abschätzen zu können. Für diesen gibt es sehr viele verschiedene Einsatzmöglichkeiten. Zum Beispiel verwendet [Elk03] diesen Ansatz zur Beschleunigung des $k$-Means-Clusterverfahrens. In [PHB+10] wird ein Datenverdichtungsverfahren vorgestellt, das mit Hilfe des Pivot-Ansatzes ein sehr schnell zu berechnendes strukturiertes Sample erstellt, auf welches anschließend Average-Linkage-SAHN-Clustering angewandt wird. Auf einige weitere mögliche Verwendungen des Pivot-Ansatzes wird im Verlauf dieses Abschnittes noch näher eingegangen. Allen gemeinsam ist die Abschätzung der Distanzen zwischen zwei beliebigen Objekten $x_i$ und $x_j$ durch die berechneten exakten Distanzen dieser Objekte zu einer Menge von Pivot-Objekten $P$. Durch Umformen der Dreiecksungleichung und Ausnutzen der Symmetrieeigenschaft kann für jeden einzelnen Pivot $x_p \in P$ eine untere Schranke der Distanz zwischen den Objekten $x_i$ und $x_j$ berechnet werden:

$$d(x_p, x_i) \leq d(x_p, x_j) + d(x_j, x_i) \tag{4.12}$$

$$\Leftrightarrow d(x_p, x_i) - d(x_p, x_j) \leq d(x_j, x_i) \tag{4.13}$$

$$\Rightarrow |d(x_p, x_i) - d(x_p, x_j)| \leq d(x_j, x_i) \quad | \text{ Symmetrie} \tag{4.14}$$

Im einfachsten Fall können diese unteren Schranken dazu verwendet werden, exakte Distanzkalkulationen einzusparen. Bei der Nächste-Nachbar-Suche können z. B. exakte Distanzberechnungen zu Objekten, deren untere Schranke für die Distanz größer ist als die Distanz zu dem Objekt mit aktuell minimaler Distanz, eingespart werden. Für alle anderen Objekte muss die exakte Distanz berechnet werden. In [Elk03] werden z. B. $k$-Means-Clusterzentroiden als Pivots verwendet, um die Distanzberechnungen zu einigen Clusterzentroiden zu sparen. Diese einfachen Ansätze verbessern die Laufzeit für den Fall, dass die $|P|$ Berechnungen für die untere Schranke schneller sind als eine exakte Distanzberechnung. Der Nutzen dieser Technik ist daher stark abhängig von der Rechenkomplexität des verwendeten Distanzmaßes. Asymptotisch ändert sich die Laufzeit der Algorithmen nicht.

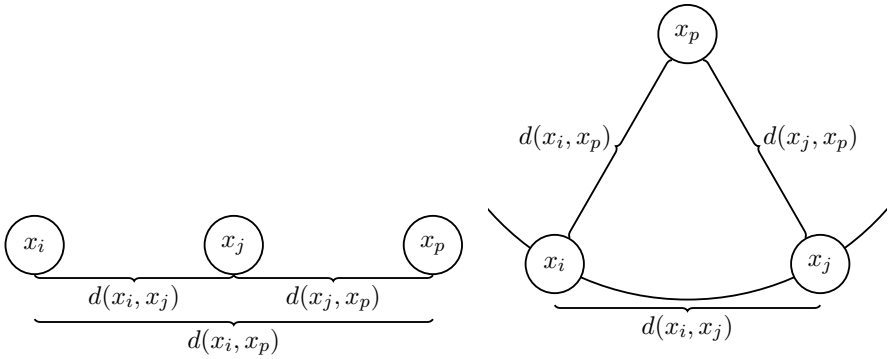
Die beste untere Schranke aller Pivots kann auch als heuristische Schätzung für die Distanzen verwendet werden:

$$\tilde{d}(x_i, x_j) = \max_{x_p \in P} |d(x_p, x_i) - d(x_p, x_j)| \tag{4.15}$$

In diesem Fall führt die Verwendung des Pivot-Ansatzes, als Alternative zu einer quadratischen Distanzmatrix (etwa beim SAHN-Clustering), zu einer Reduzierung des Speicherbedarfes von $\mathcal{O}(n^2)$ auf $\mathcal{O}(|P|\ n)$, da nur noch die Distanzen zu den Pivots gespeichert werden müssen.

Die Güte der Abschätzung hängt von der Lage des Pivots im Verhältnis zur Lage der Objekte $x_i$ und $x_j$ ab. Im euklidischen Vektorraum lässt sich dieser Zusammenhang sehr gut erläutern, weshalb er im Folgenden beispielhaft verwendet wird. Die Erkenntnisse lassen

sich aber auf jedes andere metrische Distanzmaß übertragen. Im besten Fall liegen $x_i$, $x_j$ und $x_p$ auf einer Linie und $x_p$ ist einer der beiden äußeren Punkte (vgl. Abbildung 4.8). Die geschätzte Distanz ist in diesem Fall exakt. Im schlechtesten Fall liegen $x_i$ und $x_j$ auf einer Sphäre um den Punkt $x_p$ und die Distanz wird, unabhängig von der realen Distanz, mit 0 abgeschätzt. Deshalb wird in der Praxis die beste untere Schranke aus einer Menge von Pivots $P$ verwendet.




(a) Bester Fall (b) Schlechtester Fall

**Abbildung 4.8.:** Distanzabschätzung mit Pivot

Aus dieser Beobachtung leitet sich ab, dass der Pivot-Ansatz nicht unabhängig von der Dimensionalität der Daten ist, denn mit der Dimension steigt auch die Anzahl der möglichen Fälle, bei denen die Abschätzung der Distanz ungenau ist.

Nehmen wir als Beispiel einen zweidimensionalen euklidischen Vektorraum an. In diesem Raum besitzen Pivots Kreise mit beliebigen Radien als Gebiete, in denen der schlechteste Fall der Abschätzung eintreten kann. Bei Verwendung von zwei unterschiedlichen Pivots besitzen ein Kreis um den ersten Pivot und ein zweiter Kreis um den zweiten Pivot maximal zwei Schnittpunkte. Der schlimmste Fall der Abschätzung kann also nur noch für zwei Objekte gelten, die genau auf diesen Schnittpunkten liegen. Da jedoch Kreise mit beliebigen Radien verwendet werden können, würden alle Punktepaare, die an einer Geraden welche durch den ersten und zweiten Pivot verläuft gespiegelt sind, mit einer Distanz von 0 abgeschätzt. Bereits mit Hilfe eines dritten Pivots, der nicht auf dieser Geraden liegt, wird der schlimmste Fall komplett eliminiert (vgl. Abbildung 4.9), denn es existieren keine zwei Punkte, die auf allen drei Kreisen dieser unterschiedlicher Pivots liegen.

Das Beispiel lässt sich auch für höhere Dimensionen erweitern. Im dreidimensionalen Vektorraum sind die Schnitte zwischen zwei Kugeln Kreise. Daher wird ein Pivot mehr benötigt, um den schlechtesten Fall der Abschätzung auszuschließen. Der vierte Pivot dürfte in diesem Fall nicht auf der Ebene liegen, welche die drei anderen Pivots aufspannen. Für einen Datensatz mit einer Dimensionalität $d$ müssen also mindestens $d+1$ Pivots verwendet werden, um den schlimmsten Fall der Abschätzung auszuschließen.

Für die erwartete Güte der Abschätzung zweier Objekte $x_i$ und $x_j$ ist jedoch nicht nur der Ausschluss des schlechtesten Falles entscheidend. Daher müssen im Allgemeinen deutlich mehr als $d+1$ Pivots gewählt werden. Wie viele Pivots tatsächlich gewählt werden müssen, kann an dieser Stelle nicht abschließend bestimmt werden und hängt von weiteren Aspekten des Datensatzes ab.

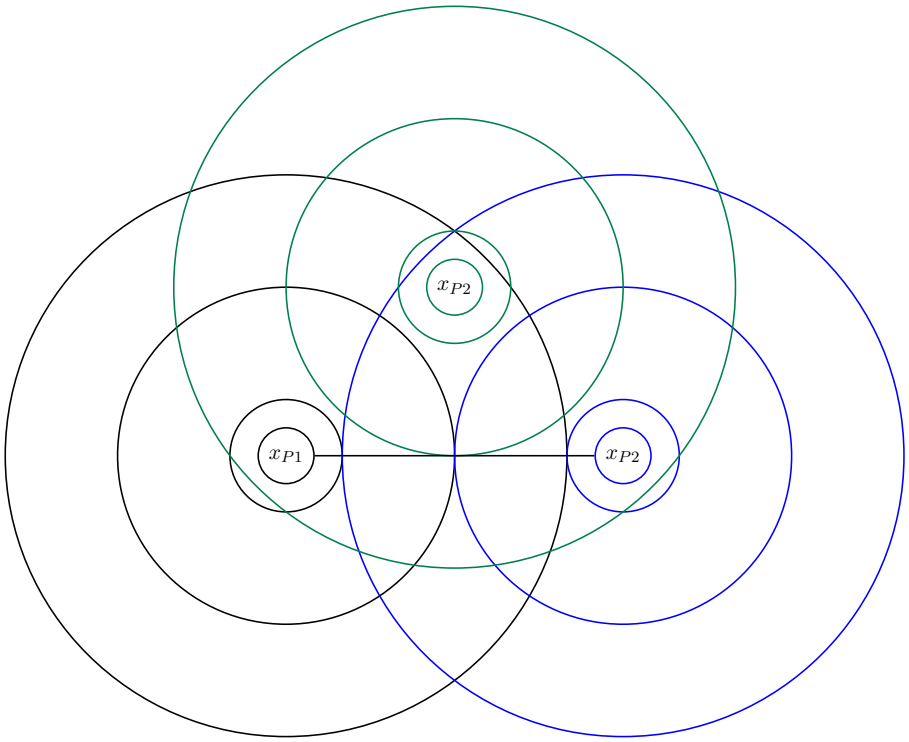



**Abbildung 4.9.:** Einfluss der Pivotanzahl auf die Qualität der Abschätzung (zweidimensional, euklidisch)

Einer dieser Aspekte ist, dass die Anzahl der Merkmale, welche für das euklidische Distanzmaß verwendet werden, nicht unbedingt mit der intrinsischen Dimensionalität der Daten übereinstimmen muss. Es kann z.B. vorkommen, dass die Daten alle auf einer Hyperebene im Raum liegen. Die ausschlaggebende Dimensionalität ist dann die Dimensionalität der Hyperebene und nicht die Dimensionalität des Raumes, in den sie eingebettet ist. Die intrinsische Dimensionalität $\rho$ wird in [CN01] durch die folgende Formel berechnet ($\mu$ bezeichnet die durchschnittliche Distanz und $\sigma$ die Standardabweichung):

$$\rho = \frac{\mu^2}{2\ \sigma^2} \tag{4.16}$$

Die Effektivität der Pivots hängt daher von der (intrinsischen) Dimensionalität und der Lage der Objekte relativ zu den Pivots ab. In [BNC03] wird ein Effektivitätskriterium vorgeschlagen, das all diese Aspekte berücksichtigt.

$$E_P = \mu(\tilde{d}_P) \tag{4.17}$$

$E_P$ bezeichnet daher die durchschnittliche geschätzte Distanz zwischen zwei beliebigen Objekten für eine Menge Pivots $P$. Wenn dieser Wert für eine Auswahl von Pivots $P$ größer ist als für eine andere $P'$, dann lässt sich daraus schließen, dass die Distanzabschätzungen durch $P$ exakter sind als die durch $P'$. Eine Auswahl von Pivots mit großem Mittelwert ist einer Auswahl von Pivots mit kleinem Mittelwert vorzuziehen. Das Kriterium eignet sich daher nur zum Vergleich der Effektivität zweier Pivotmengen, für einen bestimmten Datensatz, und stellt kein absolutes Maß dar.

Abschließend lässt sich zusammenfassen, dass der Pivot-Ansatz nicht unabhängig von der intrinsischen Dimensionalität der Daten ist. Deshalb steigt die Anzahl der benötigten Pivots mit der Anzahl der (intrinsischen) Dimensionen, falls eine gleichbleibende, erwartete Qualität erforderlich ist. Wie hoch dieser Zuwachs ist, hängt jedoch auch von den Daten und der Wahl der Pivots ab.

#### 4.2.3.1. Pivot-Baum

Wenn der Pivot-Ansatz verwendet wird, um die Nächste-Nachbar-Suche zu beschleunigen, dann ist die Güte der Schätzung vor allem für kleine Distanzen wichtig, denn große Distanzen werden unabhängig von einem relativ großen Fehler als nächster Nachbar mit einer hohen Wahrscheinlichkeit verworfen. Ist die Distanz zu nahe liegenden Objekten jedoch ungenau, werden entweder mehr exakte Distanzberechnungen benötigt oder die Fehlerrate steigt (im heuristischen Fall) an. Deshalb wird in [ZS06] eine Pivot-Baum-Datenstruktur vorgeschlagen. In dieser Datenstruktur werden nicht alle Pivots für alle Distanzabschätzungen herangezogen. Einige Pivots werden nur für lokale Abschätzungen benutzt und erhöhen damit die Präzision für kurze Distanzen.

Im Folgenden wird eine, im Rahmen dieser Arbeit modifizierte, Variante des Pivot-Baumes beschrieben, welche eine verständlichere Notation verwendet und die Baumstruktur modifiziert, um einige unnötig komplizierte Zusammenhänge zu eliminieren. Durch die Modifikation der Baumstruktur ändert sich die Semantik oder das Laufzeitverhalten des Pivot-Baumes nicht.

Der Pivot-Baum ist ein gewurzelter Baum. Jedem Knoten $N_i$ ist eine Menge von Objekten $X_i$ und eine Menge von Pivots $P_i$ mit $|P_i| = f$ zugeordnet. $P_i$ wird mit Hilfe von zufälligem Sampling aus der Menge $X_i$ ausgewählt. Für die Wurzel $N_W$ gilt $X_W = \mathcal{X}$.

Bei der Konstruktion des Baumes wird durch *Split*-Operationen immer das Blatt $N_{\max}$ mit der maximalen Anzahl von Objekten um $f$ neue Kinder erweitert.

$$N_{\max} = \text{argmax}_{N_i}(|X_i|) \tag{4.18}$$

Für jeden Pivot $x_p \in P_{\max}$ wird ein neuer Baumknoten $N_{x_p}$ erzeugt. Die Objekte $x_i \in X_{\max}$ sind genau dann Element der Menge $X_{x_p}$, wenn gilt: $\forall x_{p'} \in P \setminus \{x_p\} : d(x_p, x_i) \leq d(x_{p'}, x_i)$. Knoten mit $|X| \leq 1$ werden verworfen. Das Verfahren wird gestoppt, wenn eine festgelegte Anzahl an Blättern erreicht ist.

Für die Schätzung der Distanzen zwischen zwei Objekten $x_i$ und $x_j$ werden nun immer nur die gemeinsamen Pivots $P_{i\cup j}$ herangezogen:

$$P_{i\cup j} = \{x_p \in P_k \mid x_i, x_j \in X_k\} \tag{4.19}$$

$$\tilde{d}(x_i, x_j) = \max_{x_p \in P_{i\cup j}} |d(x_p, x_i) - d(x_p, x_j)| \tag{4.20}$$

Je tiefer ein Knoten $N_i$ im Baum liegt, desto kleiner ist die Menge $X_i$. Die entsprechenden Pivots $P_i$ werden ausschließlich für Distanzberechnungen verwendet, bei denen beide Objekte in der Menge $X_i$ liegen. Diese Objekte sind dem entsprechen Pivot des Elternknoten nahe und daher kann von einer gewissen Lokalität der Menge $X_i$ gesprochen werden. Für die Berechnung der Distanzen zwischen lokalen Objekten werden daher mehr Pivots verwendet als für weit entfernte. Somit kann die Anzahl der benötigten Pivots für eine Nächste-Nachbar-Suche gering gehalten werden, ohne die Qualität der Abschätzung negativ zu beeinflussen.

#### 4.2.3.2. Best-Frontier-Suche

In [Zho09] wird eine Methode beschrieben, mit deren Hilfe die Nächste-Nachbar-Suche beschleunigt werden kann. Voraussetzung ist die Verwendung des Pivot-Ansatzes. Das Verfahren ist heuristisch und ermittelt den nächsten Nachbar mit Hilfe der Distanzschätzung des Pivot-Ansatzes:

$$\tilde{NN}(x_i) = \text{argmin}_{x_j}(\tilde{d}(x_i, x_j)) \tag{4.21}$$

Für die Best-Frontier-Suche wird eine Datenstruktur benötigt, deren Aufbau in Abbildung 4.10 festgehalten ist. Jedes Objekt $x \in \mathcal{X}$ wird für jeden Pivot $x_{p_i} \in P$ in einer sortierten Distanzliste $L_i$ gespeichert. Die Sortierung der Objekte in $L_i$ richtet sich nach der Distanz $d(x, x_{p_i})$. Durch die Verkettung (mittels Zeigern) gleicher Objekte zwischen allen Distanzlisten kann die Position von Objekten aus einer Liste in allen anderen Listen effizient gefunden werden. Das erste Objekt kann mit Hilfe von Arrays (bei indexierten Clustern) oder einer Hashing-Datenstruktur ebenfalls in konstanter Zeit gefunden werden.

Nehmen wir im Folgenden an, dass der nächste Nachbar von $x_i$ gesucht werden soll. Bei der Betrachtung einer einzelnen sortierten Liste $L_i$ fällt auf, dass die durch den Pivot $x_{p_i}$ geschätzte Distanz besonders klein für ein Objekt $x_j$ ist, welches besonders nahe bei $x_i$ steht. Eines der beiden benachbarten Objekte von $x_i$ in $L_i$ besitzt die kleinste untere Schranke für die Distanz zu $x_i$. In jede Richtung der sortierten Distanzliste $L_i$ steigt diese Schranke mit zunehmendem Abstand zu $x_i$ monoton an, denn es gilt:

$$\tilde{d}_{x_{p_i}}(x_i, x_j) = |d(x_{p_i}, x_i) - d(x_{p_i}, x_j)| \tag{4.22}$$

Da die untere Schranke von einem einzigen Pivot als Abschätzung nicht ausreichend genau ist (vgl. Argumentation Abschnitt 4.2.3) werden jedoch mehrere Pivots verwendet. Daher

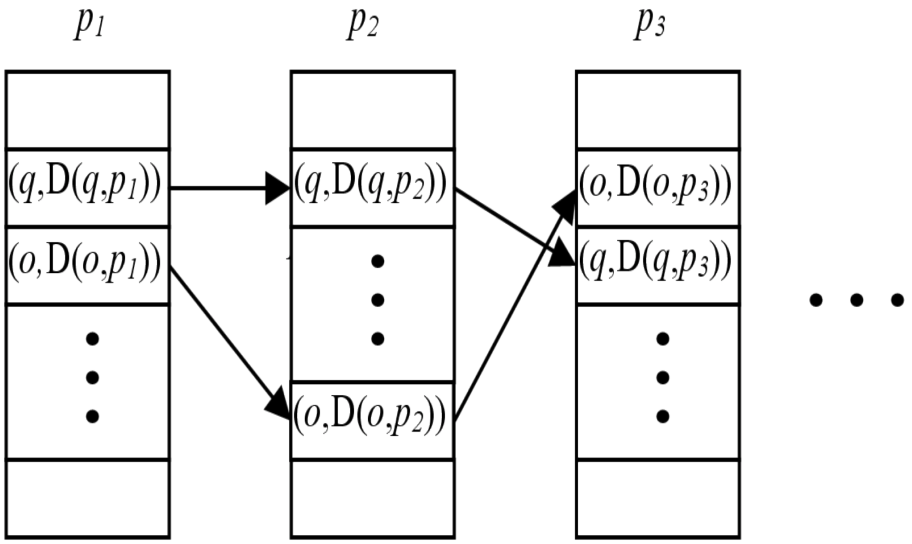



**Abbildung 4.10.:** Datenstruktur für die Best-Frontier-Suche ($q, o \in \mathcal{X}; p_i \in P$), Quelle: [Zho09]

muss die beste untere Schranke aller Pivots für ein Objekt verwendet werden. Ein Objekt $x$, welches ein Nachbar von $x_i$ in einer Liste $L$ ist, kann in einer anderen Liste $L'$ beliebig weit von $x_i$ entfernt liegen. Um die beste Abschätzung $\tilde{d}(x, x_i)$ zu ermitteln, muss $x$ in jeder Liste gefunden werden. Als Lösung dieses Problems wird eine Best-Frontier-Suche vorgeschlagen.

Die Frontier ist eine Prioritätswarteschlange $F$, die Tupel aus Objekten und Pivots enthält. Sie enthält zum Zeitpunkt der Initialisierung ein Tupel $(x_j, x_{p_i})$ pro Nachbarobjekt $x_j$ von $x_i$ aus jeder Distanzliste $L_i$. Um das Objekt $x_i$ schnell in jeder Liste finden zu können, wurden die Objekte miteinander verkettet. Sortiert wird $F$ nach der geschätzten Distanz $\tilde{d}_{x_{p_i}}(x_i, x_j)$.

Beim Ausführen der Best-Frontier-Suche wird immer das kleinste Element $(x_j, x_{p_i})$ aus $F$ entfernt und es wird dabei für jedes Objekt $x_j$ gezählt wie häufig es aus $F$ entfernt wird. Nach jedem Entfernen wird ein neues Tupel $(x_k, x_{p_i})$ zu $F$ hinzugefügt, wobei $x_k$ der Nachfolger von $x_j$ in $L_i$ ist. Nachfolger meint in diesem Zusammenhang den Nachbar von $x_j$ in $L_i$, der weiter von $x_i$ entfernt ist. Dieser Vorgang wird so lange wiederholt bis ein Objekt $|P|$ mal gefunden wurde. Dieses Objekt wird als nächster Nachbar ausgegeben.

Das Vorgehen lässt sich durch folgende Beobachtungen erklären. Die Distanzen $\tilde{d}_{x_{p_i}}(x_i, x_j)$ der Tupel $(x_j, x_{p_i})$, die aus $F$ entfernt werden, steigen monoton an. Dies liegt daran, dass die Distanz $\tilde{d}_{x_{p_i}}(x_i, x_k)$ für jedes im Laufe der Best-Frontier-Suche hinzugefügte Tupel $(x_k, x_{p_i})$ größer ist als die Distanz $\tilde{d}_{x_{p_i}}(x_i, x_j)$ des zuvor entfernten Tupels $(x_j, x_{p_i})$. Zudem ist $F$ nach genau diesen Distanzen sortiert. Nach $|P|$ Entfernungen eines Objektes aus $F$ ist die beste Schranke für dieses Objekt gefunden worden. Zudem werden

alle weiteren Schranken der Elemente aus $F$ durch die Monotonie nur noch größer. Damit ist der heuristische, minimale nächste Nachbar gemäß Gleichung 4.21 gefunden.

Der Aufbau der Datenstruktur bedingt das Sortieren von $|P|$ Listen der Größe $n = |\mathcal{X}|$. Die Rechenkomplexität ist daher durch $\mathcal{O}(|P|\ n\ \log(n))$ beschränkt. Für eine einzige Nächste-Nachbar-Suche würde sich der Aufwand einer Best-Frontier-Suche also nicht lohnen. Alle bekannten hierarchischen Clusterverfahren benötigen jedoch mindestens $n$ Nächste-Nachbar-Suchen.

Die Laufzeit der Best-Frontier-Suche selbst ist genau wie bei der naiven nächsten-Nachbar-Suche linear, wenn die Anzahl der Pivots $|P|$ konstant ist. Im schlimmsten Fall werden alle Objekte $|P| - 1$ mal gezählt bevor der nächste Nachbar ausgegeben wird. Damit sind $(|P|-1)\ n$ Einfüge- und Löschoperationen der Prioritätswarteschlange notwendig. Die Laufzeit der Best-Frontier-Suche beträgt damit $\mathcal{O}(|P|\ \log(|P|)\ n)$. Dem soeben skizzierten schlimmsten Fall steht im besten Fall eine Laufzeit von $\Omega(|P|\ \log(|P|))$ gegenüber. Im diesem Fall wird das selbe Objekt direkt am Anfang der Best-Frontier-Suche $|P|$ mal nacheinander aus der Warteschlange entfernt.

Die obere Schranke für die Laufzeit der Best-Frontier-Suche kann durch eine Begrenzung der Suchtiefe, auf Kosten der Qualität, verbessert werden. Wenn nach Erreichen der maximalen Suchtiefe $s$ der nächste Nachbar noch nicht gefunden wurde, wird eine weitere Heuristik angewandt. In diesem Fall wird das Objekt aus $F$, welches bisher am häufigsten gezählt wurde als nächster Nachbar ausgegeben. Die obere Schranke der Laufzeit dieser Variante beträgt daher $\mathcal{O}(|P|\ \log(|P|)\ s)$.

Interessanterweise kann die Best-Frontier-Suche in Kombination mit dem Pivot-Baum verwendet werden. In [ZS06] wird diese Kombination für die Beschleunigung von OPTICS genutzt. Als kleiner Unterschied zu der dargestellten Best-Frontier-Suche wird jedoch nicht nur der nächste Nachbar, sondern die $k$ nächsten Nachbarn gesucht. Die notwendigen Modifikationen sind jedoch minimal und die Best-Frontier-Suche muss nur so lange weitergeführt werden, bis $k$ Objekte $|P|$ mal gefunden wurden.

Um eine logarithmische Laufzeit für die Best-Frontier-Suche garantieren zu können, muss der Zusammenhang zwischen $s$ und der Qualität der Nächste-Nachbar-Suche weiter erforscht werden. Nur wenn sich die Suchtiefe bei gleichbleibender Qualität logarithmisch begrenzen lässt, kann von einer sinnvollen Heuristik mit logarithmischer Laufzeit gesprochen werden. In der zugrunde liegenden Literatur sind dazu leider nicht ausreichend viele Informationen vorhanden.

# Kapitel 5.

# Ein neues heuristisches SAHN-Clusterverfahren

Dieses Kapitel entwickelt einen neuen Ansatz für ein heuristisches SAHN-Clusterverfahren. In Abschnitt 5.1 wird eine Analyse der bekannten Beschleunigungsverfahren in Bezug auf ihre Vor- und Nachteile vorgenommen und ein Beschleunigungsverfahren ausgewählt. Auf Grundlage dieser Auswahl wird in Abschnitt 5.2 ein passendes SAHN-Clusterverfahren ermittelt. Abschnitt 5.3 beschreibt die Umsetzung der Kombination des Beschleunigungs- und Clusterverfahrens sowie notwendige Anpassungen bestehender Ansätze. Abschnitt 5.4 geht auf die Implementierung in der Software Scaffold Hunter ein.

## 5.1. Auswahl eines Beschleunigungsverfahren

Um ein geeignetes Beschleunigungsverfahren aus den bekannten Verfahren auszuwählen, ist die Zielsetzung von hoher Bedeutung. Deshalb sind die Anforderungen an dieser Stelle noch einmal aufgeführt:

- Anwendbar auf allgemeine metrische Distanzmaße. Insbesondere keine Beschränkung auf den Vektorraum oder das euklidische Distanzmaß.
- Anwendbar bei aufwändig zu berechnenden Distanzmaßen.
- Anwendbar bei mittlerer bis hoher Dimensionalität der Daten.
- Verwendung bekannter Cluster- und Linkage-Verfahren, um den Benutzern des Scaffold Hunter die Anwendung ohne zusätzliche Einarbeitung zu ermöglichen und die Akzeptanz zu erhöhen.

Die erste Anforderung führt bereits zum Ausschluss aller gitterbasierten Clusterverfahren, da diese auf den Vektorraum beschränkt sind. Alle Indexstrukturen, welche eine asymptotische Verbesserung der Laufzeit der Nächste-Nachbar-Suche zur Folge hätten, benötigen ebenfalls das euklidische Distanzmaß.

Der Pivot-Ansatz (in Kombination mit der Best-Frontier-Suche) und die Datenverdichtung bleiben daher als einzige Ansätze übrig, die eine Reduzierung der asymptotischen Laufzeit ermöglichen. Beide Verfahren sind heuristisch und weisen Verzerrungen bei den Clusterergebnissen auf.

Für die Datenverdichtungsverfahren gilt, dass die Feinstruktur der Clusterhierarchie verloren geht. Dies führt zu notwendigen Anpassungen bei der Darstellung der Ergebnisse und einer Verminderung der Aussagekraft des Dendrogramms. Die Datenverdichtungsverfahren versprechen eine sehr starke Beschleunigung bei Auswahl eines kleinen Samples. Jedoch gilt, dass die Genauigkeit sinkt je kleiner das Sample gewählt wird. Verschiedene Optimierungen der Berechnung von Distanzen zwischen verschiedenen Mikroclustern (vgl. Abschnitt 4.2.1.2 und [PHB+10]) minimieren dieses Problem, sind jedoch von dem verwendeten Linkage-Verfahren abhängig und schränken daher die Flexibilität ein.

Wie groß das Optimierungspotenzial des Pivot-Ansatzes ist, hängt im Wesentlichen von der Wahl des Parameters $s$ (maximale Suchtiefe) der Best-Frontier-Suche ab. Wünschenswert ist eine konstante oder logarithmische Größe in Bezug auf $|\mathcal{X}|$. Kann bei entsprechender Wahl von $s$ eine gleichbleibende Qualität der Heuristik gezeigt werden, hängt die Laufzeit der heuristischen Nächste-Nachbar-Suche nicht oder nur noch logarithmisch von der Eingabegröße ab. Der Zusammenhang zwischen der Größe von $s$ und der Qualität der Clusterergebnisse ist nicht bekannt. In der zugrundeliegenden Doktorarbeit [Zho09] wird für die Evaluation der Best-Frontier-Suche willkürlich ein fester Parameterwert gewählt. Die Ergebnisse werden daraufhin dem Data-Bubbles-Ansatz gegenübergestellt. Der Einfluss des Parameters $s$ auf die Qualität der Heuristik lässt sich daraus nicht ableiten. Die Wahl von $s$ ist zudem nicht unabhängig von den zugrundeliegenden Daten. Ohne eine Beschränkung ist die Suchtiefe und damit die Laufzeit der Best-Frontier-Suche von den Daten abhängig (vgl. Laufzeitanalyse in Abschnitt 4.2.3.2). Im besten Fall würde die Beschränkung der Suchtiefe deshalb gar keinen Einfluss auf das Ergebnis der Nächste-Nachbar-Suche haben, da der nächste Nachbar bereits vor Erreichen der Beschränkung gefunden wird. Eine empirische Auswertung dieses Zusammenhangs ist notwendig, um die Effektivität des Verfahrens bewerten zu können.

Ein Nachteil des Pivot-Ansatzes, in Bezug auf SAHN-Clusterverfahren, ist die Beschränkung auf Metriken als Inter-Cluster-Distanzmaß, denn das schränkt die Wahl des Linkage-Verfahrens ein. Median- und Zentroid-Linkage sind die einzigen in dieser Arbeit vorgestellten (vgl. Tabelle 2.1) metrischen Linkage-Verfahren. Die Einschränkung auf spezielle Linkage-Verfahren gilt, mit Ausnahme von zufälligem Sampling, ebenfalls für die Datenverdichtungsverfahren. Zufälliges Sampling verzerrt die Ergebnisse – im Vergleich zu den restlichen vorgestellten Datenverdichtungsverfahren – deutlich stärker und scheidet daher aus der Auswahl möglicher Verfahren aus. Die in Frage kommenden Datenverdichtungsverfahren und der Pivot-Ansatz unterscheiden sich daher, in Bezug auf die Einschränkung der Linkage-Verfahren, nicht.

Ein weiterer kritischer Punkt ist die Abhängigkeit der Pivotanzahl $|P|$ von der Dimensionalität der Daten (vgl. Abschnitt 4.2.3). Es ist jedoch zu hoffen, dass der Pivot-Baum zu einer deutlichen Reduktion der notwendigen Anzahl von Pivots führt. Die Best-Frontier-Suche wurde in Kombination mit dem Pivot-Baum bereits für das OPTICS-Clusterverfahren verwendet. Dessen Verwendung im Scaffold Hunter würde aber eine völlig neue Darstellung der Ergebnisse und für den Benutzer eine Einarbeitung in das neue Clusterverfahren benötigen. Es ist daher wünschenswert, diese Kombination auch für SAHN-Clusterverfahren anzuwenden. Dabei ergibt sich das Problem, dass die Datenstruktur des Pivot-Baumes statisch ist, ein SAHN-Clusterverfahren jedoch Cluster zu neuen Clustern verschmilzt. Da ein regelmäßiger Neuaufbau der Datenstruktur aufgrund des hohen rechnerischen Aufwands kein Sinn ergibt, muss die Datenstruktur das Löschen und Einfügen von Clustern effizient ermöglichen. Dass die Datenstruktur dahingehend modifiziert werden kann, wird in Abschnitt 5.3 gezeigt.

Die Laufzeit der beiden Verfahren ist nicht einfach miteinander vergleichbar, da sie von zu vielen Parametern abhängt. Um diese Fragestellung bewerten zu können, müsste die Qualität und die Laufzeit der beiden Verfahren für unterschiedliche Datensätze und unterschiedliche Werte der Parameter empirisch überprüft werden. Der Informationsverlust der beiden Verfahren unterscheidet sich zudem fundamental. Welcher der beiden Ansätze zu favorisieren ist, hängt entscheidend von dem Anwender und seiner aktuellen Zielsetzung ab. Unterschiedliche Verfahren bedingen zudem in vielen Fällen unterschiedliche Linkage-Verfahren, wodurch ein Qualitätsvergleich unmöglich ist. Ein direkter Vergleich der Ergebnisse ist daher nur in speziellen Fällen möglich. Die Implementierung beider Ansätze würde den Umfang dieser Diplomarbeit überschreiten.

Aufgrund der fehlenden Feinstruktur und der damit verminderten Aussagekraft der Clusterergebnisse von Datenverdichtungsverfahren habe ich mich entschieden, den Pivot-Ansatz weiter zu verfolgen. Die Klärung der Fragen in Bezug auf den Parameter $s$ und die damit verbunden Effizienz der Best-Frontier-Suche sind wichtig um Wissenslücken in Bezug auf die Best-Frontier-Suche zu füllen. Zudem ist mir kein heuristisches SAHN-Clusterverfahren für allgemeine metrische Distanzmaße mit subquadratischer Laufzeit bekannt, welches keine Datenkompression verwendet. Der mögliche wissenschaftliche Fortschritt erschient mir auf diesem Gebiet daher höher zu sein als bei den Datenverdichtungsverfahren.

## 5.2. Auswahl eines SAHN-Clusterverfahrens

Im Folgenden werden die in Abschnitt 4.1.3 vorgestellten SAHN-Clusterverfahren auf ihre Tauglichkeit in Bezug auf den Pivot-Ansatz in Kombination mit der Best-Frontier-Suche und dem Pivot-Baum untersucht.

### 5.2.1. NNChain

Der NNChain-Algorithmus benötigt die Distanzmatrix ausschließlich für die Nächste-Nachbar-Suche. Damit kann eine alternative Nächste-Nachbar-Suchstrategie eingesetzt werden und die entsprechende Datenstruktur die Distanzmatrix ersetzen. Ein Problem des NNChain-Algorithmus ist jedoch die Beschränkung auf Linkage-Verfahren, welche die Reducibility-Eigenschaft erfüllen. Beide metrischen Linkage-Verfahren erfüllen diese Eigenschaft nicht. Die Reducibility-Eigenschaft kann zudem durch heuristische Nächste-Nachbar-Suchen verletzt werden.

Durch die Verletzung der Reducibility-Eigenschaft ist nicht mehr garantiert, dass aufeinanderfolgende Kettenglieder im weiteren Verlauf des Algorithmus nächste Nachbarn bleiben. Dadurch wird die komplette Kette ungültig. Deshalb müsste man die Kette bei jeder Verschmelzung neu beginnen. Dies würde die Laufzeit auf $\mathcal{O}(n^3)$ verschlechtern. Im heuristischen Fall könnte man diese Verletzungen ignorieren, sofern sie nicht zu häufig auftreten. Jedoch können dadurch Kreise in der Kette entstehen. Diese können nicht einfach aufgelöst werden.

Nehmen wird den hypothetischen Fall an, dass der Kreis

$$\cdots \to x_i \to x_j \to \cdots \to x_k \to x_i$$

entstanden ist. Es gibt zwei mögliche Fälle für den weiteren Verlauf des Algorithmus. Im ersten Fall wird durch die Heuristik das reziproke Nächste-Nachbar-Paar nicht gefunden und der Kreis wird immer wieder durchlaufen. Dies kann geschehen wenn der nächste Nachbar von $x_i$ nicht korrekt ermittelt wird. Der echte nächste Nachbar von $x_i$ könnte das Objekt $x_k$ sein. Wenn man diesen ersten Fall effizient erkennen kann, dann ist es möglich $x_i$ und $x_k$ zu verschmelzen. Dadurch geht der restliche Kreis verloren. Im zweiten Fall ist das Kettenglied $x_i \to x_j$ inzwischen ungültig geworden und der nächste Nachbar von $x_i$ ist nun ein anderes Objekt. Dadurch geht entweder der Teil der Kette bis zu dem Element $x_i$ oder der Kreis exklusive $x_i$ verloren. Die Frage ist, wie häufig eine solche Situation auftritt und ob die Worst-Case-Analyse mit der Praxis übereinstimmt. Darüber kann ohne genauere empirische Untersuchungen keine Aussage getroffen werden.

Resümierend erscheint die Laufzeitreduzierung des NNChain-Algorithmus mittels heuristischen Nächste-Nachbar-Suchen als wenig aussichtsreich. Ohne eine Analyse über die Häufigkeit der Kreisbildung sollte dieser Algorithmus jedoch nicht vollständig verworfen werden.

### 5.2.2. Dynamic Closest Pair

Die Dynamic-Closest-Pair-Datenstruktur wurde in zwei Varianten vorgestellt (vgl. Abschnitt 4.1.3.2).

Die Variante mit quadratischer Laufzeit ist von der Struktur her ungeeignet, die Best-Frontier-Suche einzusetzen, da sie eine Indexstruktur über der quadratischen Distanzmatrix darstellt. An keiner Stelle kommt eine Nächste-Nachbar-Suche zum Einsatz, die einen Einfluss auf die Laufzeit hätte.

Die zweite Variante mit linearem Speicherbedarf arbeitet zwar intensiv mit Hilfe von Nächste-Nachbar-Suchen, um die Datenstruktur aufzubauen, verwendet für die Suche nach dem minimalen reziproken Nächste-Nachbar-Paar jedoch einen linearen Durchlauf über $\mathcal{O}(n)$ Elemente. Deshalb könnte diese Datenstruktur zwar im Aufbau beschleunigt werden, die Laufzeit eines entsprechenden Clusterverfahren würde aber mindestens $\Omega(n^2)$ betragen. Diese Laufzeit stellt keine Verbesserung gegenüber bekannten Verfahren dar.

### 5.2.3. Single-Linkage-MST-Algorithmus

Der Single-Linkage-Minimal-Spanning-Tree-Algorithmus (vgl. Abschnitt 4.1.3.4) lässt sich mit der Best-Frontier-Suche kombinieren. Mit Hilfe der Best-Frontier-Suche kann ein $k$-NN Graph, bei logarithmischer maximaler Suchtiefe $s$, in $\mathcal{O}(n\ \log(n))$ konstruiert werden. Ein $k$-NN Graph $G_k = (V, E_k)$ entspricht einem Subgraphen des vollständigen Distanzgraphen $G = (V, E)$ mit $E_k \subseteq E$. In $E_k$ sind nur die Kanten ($x$,$y \in NN_k(x)$) enthalten, welche durch die k-nächste Nachbarschaft $NN_k$ eines Knotens induziert werden. Für den heuristischen Fall kann der MST-Algorithmus auf diesem reduzierten Graphen $G_k$ ausgeführt werden. Die Laufzeit der Berechnung des minimalen Spannbaumes bei konstanter Wahl von $k$ liegt bei $\mathcal{O}(n\ \log(n))$ (Algorithmus von Kruskal oder Prim). Damit ist auch die Gesamtlaufzeit durch $\mathcal{O}(n\ \log(n))$ beschränkt.

### 5.2.4. Generic Clustering (Müllner)

Der Generic-Clustering-Algorithmus eignet sich für die metrischen Linkage-Verfahren. In einer modifizierten Variante (vgl. Abschnitt 5.3.1) lässt sich die Laufzeit auf $\mathcal{O}(n^2\ (\log(n)+k))$ und $\Omega(n\ (\log(n)+k))$ senken, wenn $k$ die Laufzeit der Nächste-Nachbar-Suche darstellt. Wenn die Berechnung eines nächsten Nachbarn in sublinearer Zeit möglich ist, reduziert sich die Gesamtlaufzeit des Algorithmus daher gegenüber dem ursprünglichen Generic-Clustering-Algorithmus.

Der Algorithmus eignet sich deswegen gut, um ihn mit der Best-Frontier-Suche zu kombinieren. Es ist zu hoffen, dass sich die praktische Laufzeit weiterhin an der unteren Schranke der theoretischen Laufzeit orientiert.

### 5.2.5. Fazit

Der Single-Linkage-MST-Algorithmus und der Generic-Clustering-Algorithmus eignen sich gut, um die Best-Frontier-Suche zu integrieren. Das Single Linkage-Verfahren leidet jedoch unter dem so genannten *Chaining-Effekt*. Bei verrauschten Datensätzen können sich schnell

Ketten zwischen unterschiedlichen Clustern bilden, durch die die Trennung dieser Cluster verloren geht. In der Molekularbiologie und der Chemie haben sich vor allem Linkage-Verfahren durchgesetzt, die relativ kompakte Cluster erzeugen (vgl. [DB03]). Aus diesem Grund wird der Ansatz zur Beschleunigung des Single-Linkage-MST-Algorithmus nicht weiter verfolgt. Der Single-Linkage-MST-Algorithmus ist aber eine interessante Alternative, die nach Möglichkeit außerhalb dieser Diplomarbeit erforscht werden sollte.

Auch wenn sich die Eignung des NNChain-Algorithmus nicht ausschließen lässt, wird er aufgrund der Kreisbildung und der dadurch entstehenden Laufzeitprobleme nicht weiter als Lösung in Betracht gezogen.

Aus diesen Gründen wird der Generic-Clustering-Algorithmus für das heuristische SAHN-Clusterverfahren verwendet.

## 5.3. Umsetzung

Dieser Abschnitt beschäftigt sich mit der Umsetzung des Generic-Clustering-Algorithmus in Kombination mit der Best-Frontier-Suche und dem Pivot-Baum.

### 5.3.1. Anpassungen am Generic-Clustering-Algorithmus

Algorithmus 5.1 stellt eine modifizierte Fassung des Generic-Clustering-Algorithmus (Algorithmus 4.1) dar, bei der die Komplexität der Nächste-Nachbar-Suche $NN(x)$ sowohl die obere als auch die untere Schranke der Laufzeit beeinflusst. Die Gesamtlaufzeit beträgt $\mathcal{O}(n^2\ (\log(n)+k))$ und $\Omega(n\ (\log(n)+k))$, wenn $k$ die Laufzeit der Nächste-Nachbar-Suche ist. Liegt die Laufzeit der Nächste-Nachbar-Suche asymptotisch unterhalb von $\mathcal{O}(\log(n))$ (z. B. $\mathcal{O}(1)$), dann hat dies keinen weiteren Einfluss auf die asymptotische Gesamtlaufzeit, da die Einfügeoperationen der Prioritätswarteschlange in diesem Fall die Laufzeit dominieren. Es bleibt festzustellen, ob sich die Laufzeit des modifizierten Algorithmus in der Praxis weiterhin an der unteren Schranke orientiert.

Die Schleifen aus Zeile 28 und 33 des ursprünglichen Algorithmus (siehe Algorithmus 4.1) sind dafür verantwortlich, dass er, unabhängig von der Komplexität der Nächste-Nachbar-Suche, eine untere Schranke von $\Omega(n^2)$ für die Laufzeit besitzt.

Im ursprünglichen Algorithmus konnte durch die Vorwärtssuche der Fall auftreten, dass ein Objektpaar $(C_k, C_{i\cup j})$ mit einer zu niedrigen Priorität in der Warteschlange einsortiert war, wenn die Reducibility-Eigenschaft für das verwendete Linkage-Verfahren nicht erfüllt ist (vgl. Argumentation über den Zweck von Schleife 33 in Abschnitt 4.1.3.3). Im Gegensatz zur ursprünglichen Variante könnte daher auf die Vorwärtssuche verzichtet werden, damit die Schleife aus Zeile 33 entfällt. Dadurch würde die Laufzeit der exakten Nächste-Nachbar-Suche verdoppelt. Das Problem mit den falschen Prioritäten tritt jedoch erst auf, wenn Cluster verschmolzen werden und der Index von $C_k$ kleiner als der von $C_{i\cup j}$ ist. Daher kann für die Initialisierung von $Q$ weiterhin die Vorwärtssuche verwendet werden.

```
1: function genericClustering(𝒳)
2:     currentLevel ← 𝒳                                  ▷ Cluster des aktuellen Levels
3:     for all x ∈ 𝒳 do                                  ▷ Initialisierung Q
4:         Q.insert(x, NN(x))          ▷ Q ist sortiert nach dist(x, NN(x)) (Wert zum
                                       Zeitpunkt der Einfügeoperation)
5:     end for
6:     while currentLevel.size() > 1 do                  ▷ Hauptschleife
7:         (x, y) ← Q.extractMin()
8:         while !currentLevel.contains(x) or !currentLevel.contains(y) do
9:                                     ▷ ungültiger Eintrag → Neuberechnung des NN
10:            if currentLevel.contains(x) then
11:                Q.insert(x, NN(x))
12:            end if
13:            (x, y) ← Q.extractMin()
14:        end while
15:        z ← mergeCluster(x,y)          ▷ (x, y) ist minimales reziprokes NN Paar
16:        currentLevel.remove(x)
17:        currentLevel.remove(y)
18:        currentLevel.add(z)
19:        Q.insert(z, NN(z))
20:    end while
21:    return currentLevel.get(0)
22: end function
```

**Algorithmus 5.1:** Modifizierter Generic-Clustering-Algorithmus

Durch die nicht erfüllte Reducibility-Eigenschaft kann es bei den Verschmelzungen vorkommen, das der nächste Nachbar beliebiger Cluster $\{C_{k_1}, \ldots, C_{k_m}\}$, welche vorher nicht $C_i$ oder $C_j$ als nächsten Nachbarn hatten, nun den Cluster $C_{i\cup j}$ als nächsten Nachbarn besitzen. Werden neu verschmolzene Cluster $C_{i\cup j}$ immer mit kleinstem Index eingefügt, dann kann die Vorwärtssuche weiterhin für alle Nächste-Nachbar-Suchen verwendet werden, denn die anschließende Nächste-Nachbar-Suche $NN(C_{i\cup j})$ wird immer über alle Cluster durchgeführt (Vorwärtssuche ist praktisch deaktiviert). Das hat zur Folge, dass die minimale Distanz aller Distanzen $D(C_k,C_{i\cup j})$ mit $C_k \in \{C_{k_1}, \ldots, C_{k_m}\}$ weiterhin mit korrekter Priorität in $Q$ einsortiert ist oder der nächste Nachbar von $C_{i\cup j}$ ein völlig anderer Cluster ist.

Es ist damit garantiert, dass in Zukunft $C_{i\cup j}$ eine geringere Priorität als $C_k \in \{C_{k_1}, \ldots, C_{k_m}\}$ besitzt und daher das minimale reziproke Nächste-Nachbar-Paar weiterhin die höchste Priorität in der Prioritätswarteschlangen besitzt. Für alle Objekte

$\{C_{k_1}, \ldots, C_{k_m}\}$, welche nun mit zu niedriger Priorität in $Q$ einsortiert sind, gilt dass sie erst dann verschmolzen werden wenn einer der beiden folgenden Fälle eintritt:

1. Der Cluster $C_{i \cup j}$ wird zu $C_{i \cup j \cup l}$ verschmolzen: Es kann nun vorkommen, dass die Nächste-Nachbar-Distanz von $C_{i \cup j \cup l}$ weiterhin kleiner ist als jene von $C_k \in \{C_{k_1}, \ldots, C_{k_m}\}$ bevor $C_{i \cup j}$ gebildet wurde. In diesem Fall hat $C_{i \cup j \cup l}$ eine niedrigere Priorität als $C_k$ und erfüllt damit die Rolle welche $C_{i \cup j}$ vorher eingenommen hat. Falls die Nächste-Nachbar-Distanz von $C_{i \cup j \cup l}$ nun aber größer ist, dann ist der alte nächste Nachbar von $C_k$ wieder der Korrekte und auch die Priorität in $Q$ ist wieder korrekt.

2. Ein Cluster aus $\{C_{k_1}, \ldots, C_{k_m}\}$ wird verschmolzen: In diesem Fall wird für den neu verschmolzenen Cluster die Nächste-Nachbar-Suche über alle andere Cluster ausgeführt, da er den kleinsten Index zugewiesen bekommt (s. o.) und der Cluster die korrekte Priorität in $Q$ hat.

Es wurde gezeigt, dass die Vorwärtssuche weiterhin korrekte Ergebnisse produziert, wenn neu verschmolzene Cluster mit minimalem Index eingefügt werden. Unabhängig davon lässt sich die Best-Frontier-Suche nicht mit einer Vorwärtssuche kombinieren. Daher bietet die Vorwärtssuche nur für die exakte Nächste-Nachbar-Suche einen Laufzeitvorteil.

Der Verzicht auf die Vorwärtssuche bedeutet für den heuristischen Fall nicht nur den Verlust einer Laufzeitoptimierung. In der ursprünglichen Variante des Algorithmus wird angenommen, dass das minimale reziproke Nächste-Nachbar-Paar auch dann in der Prioritätswarteschlange das minimale Element ist, wenn nur die eine der beiden Nächste-Nachbar-Beziehungen gefunden wurde. Für symmetrische Distanzmaße ist diese Annahme korrekt. Im heuristischen Fall kann es jedoch vorkommen, dass die minimale Nächste-Nachbar-Beziehung nicht reziprok ist, wenn mindestens eine Nächste-Nachbar-Suche fehlerbehaftet ist. Deshalb kann durch das Verzichten auf die Vorwärtssuche eine einseitig falsche Nächste-Nachbar-Suche korrigiert werden.

Unabhängig von der veränderten Indizierung neu verschmolzener Cluster bei der Vorwärtssuche hatte die Ersetzungsstrategie des ursprünglichen Algorithmus in der Schleife in Zeile 28 zum Ziel, einige Nächste-Nachbar-Suchen zu sparen (vgl. Abschnitt 4.1.3.3). Dieser Effekt wirkt sich jedoch nur bei Single- und Complete-Linkage positiv auf die Laufzeit aus, da sich für diese Linkage-Verfahren die Distanzen nur in etwa der Hälfte der Fälle ändern. Dies liegt an der Berechnung der Distanz von einem beliebigen Cluster $C_k$ zu dem neuen verschmolzene Cluster $C_{i \cup j}$ aus dem Minimum oder Maximum der Distanzen $D(C_k, C_i)$ und $D(C_k, C_j)$. Für Median- oder Zentroid-Linkage bleiben die Distanzen nur in Ausnahmefällen gleich und der Aufwand der Schleife übersteigt ihren Nutzen. Diese Ersetzungsstrategie verspricht folglich keinen Vorteil für den besprochenen Anwendungszweck und kann daher ersatzlos gestrichen werden.

Abgesehen von den oben beschriebenen Veränderungen wurden keine weiteren Anpassungen am Generic-Clustering-Algorithmus vorgenommen. Lediglich die exakte Nächste-Nachbar-Suche wird durch die Best-Frontier-Suche ersetzt.

### 5.3.2. Dynamischer Pivot-Baum für Best-Frontier-Suche

Die in Abschnitt 4.2.3.1 vorgestellte Datenstruktur wird für das heuristische SAHN-Clusterverfahren mit der Best-Frontier-Suche kombiniert. Diese Kombination ist bereits in [Zho09] für das OPTICS-Clusterverfahren verwendet worden. Die in dieser Arbeit umgesetzte Methode weicht jedoch in einigen Punkten von der ursprünglichen ab.

Für die Verwendung der Datenstruktur in Kombination mit dem SAHN-Clusterverfahren werden zudem Cluster und keine Objekte gespeichert. Diese Änderung hat jedoch keinen Effekt auf die Datenstruktur, sondern ändert nur die Bezeichnung von Elementen innerhalb der Datenstruktur.

#### 5.3.2.1. Anpassungen an der Datenstruktur

Zu jedem Pivot im Pivot-Baum werden sortierte Distanzlisten gespeichert, um die Best-Frontier-Suche zu unterstützen. Im Pivot-Baum funktioniert die Verzeigerung der Cluster zwischen den einzelnen Distanzlisten nicht mehr so einfach, wie bei der Best-Frontier-Suche ohne Pivot-Baum, da für den Start der Frontier bei einer Nächste-Nachbar-Suche $NN(C)$, der Cluster $C$ nur in einer Teilmenge der Distanzlisten gefunden werden muss. Es müsste daher unterschiedliche *Einstiegspunkte* für diese Verzeigerung geben und die Implementierung würde unnötig kompliziert. Aus diesem Grund wird die Verzeigerung nicht verwendet und durch ein Hashing ersetzt. Mit Hilfe der Hashfunktionen kann die Position von $C$ in einer Distanzliste weiterhin in konstanter Zeit gefunden werden. Alternativ kann binäre Suche verwendet werden. In diesem Fall wird eine logarithmische Laufzeit benötigt um $C$ zu finden.

Die ursprüngliche Best-Frontier-Datenstruktur ist statisch und unterstützt kein Verschmelzen von Clustern, was bei der Verwendung in Kombination mit SAHN-Clusterverfahren notwendig ist. Ein kompletter Neuaufbau der Datenstruktur für jede der $n$ Verschmelzungen kommt aufgrund des hohen Rechenaufwands jedoch nicht in Frage. Daher wird im Folgenden eine Lösch- und eine Einfügeoperation für den dynamischen Pivot-Baum beschrieben. Eine Verschmelzung kann durch zwei Löschoperationen und eine Einfügeoperation realisiert werden. Die Laufzeit der Einfügeoperation ist durch $\mathcal{O}(\log(|P|)\ \log(n))$ und die Laufzeit der Löschoperation durch $\mathcal{O}(\log(|P|))$ beschränkt.

**Löschen** Das Löschen eines Clusters $C$ im dynamischen Pivot-Baum wird durch das Löschen von $C$ in den Distanzlisten realisiert. Der Cluster $C$ ist jedoch nicht in allen Distanzlisten enthalten. Genauer ausgedrückt ist $C$ ein Element aller Baumknoten, die auf dem

direkten Pfad eines bestimmten Blattknotens $N_b$ zur Wurzel liegen (vgl. Abschnitt 4.2.3.1). Daher wird in einer Hashing-Datenstruktur eine Abbildung von jedem Cluster zum entsprechenden Blattknoten gespeichert ($N$ bezeichnet die Menge aller Baumknoten):

$$Blatt : \mathcal{C} \rightarrow N \tag{5.1}$$

Anschließend können alle relevanten Knoten und Pivots bzw. Distanzlisten durch Traversierung der Eltern-Beziehung gefunden werden.

Die Laufzeit der Löschfunktion hängt damit von der Tiefe des Pivot-Baumes ab. Welche Tiefe der Pivot-Baum im Verhältnis zur maximalen Blattanzahl besitzt, hängt von der Selektionsstrategie der Split-Operation ab. In der zugrundeliegenden Literatur wird immer das Blatt mit der größten Anzahl von Objekten gewählt. Ein durch diese Strategie erstellter Baum besitzt zwar eine erwartete, aber nicht garantierte logarithmische Tiefe. Durch die Verwendung einer alternativen Blattselektionsstrategie, welche das Blatt mit der kleinsten Distanz zur Wurzel wählt, kann eine logarithmische Tiefe garantiert werden. Die ursprüngliche Strategie löst jedoch dichte Regionen feiner auf und steigert dadurch die Effektivität der Pivots. Daher wird sie auch weiterhin verwendet.

Falls Hashing für die Distanzlisten verwendet wird, kann $C$ in konstanter Zeit pro Distanzliste gefunden werden. Dadurch ergibt sich eine Laufzeit für die Löschoperation von $\mathcal{O}(f\ \log_f(|P|))$, falls der Baum eine logarithmische Tiefe besitzt. Die Variable $f$ bezeichnet (s. a. Abschnitt 4.2.3.1) die Pivotanzahl pro Knoten.

Im Fall von binärer Suche verschlechtert sich die Laufzeit auf $\mathcal{O}(f\ \log_f(|P|)\ \log(n))$. Diese Verschlechterung ist jedoch für die asymptotische Gesamtlaufzeit irrelevant, da (für die Verschmelzung zweier Cluster) immer zwei Löschoperationen zusammen mit einer Einfügeoperation ausgeführt werden. Mit Hilfe von amortisierter Analyse können die Kosten der Löschoperationen daher auf die Einfügeoperation umgelegt werden. Der Parameter $f$ wird im Folgenden als konstant angenommen, da er unabhängig von der Größe des Datensatzes ist.

**Einfügen** Das Einfügen eines Clusters $C$ in den dynamischen Pivot-Baum funktioniert nach dem selben Prinzip wie das Löschen. Es gibt jedoch zwei Unterschiede:

Zum einen kann das Finden der Position, an welcher $C$ in die Distanzliste eingefügt werden soll, nicht durch Hashing beschleunigt werden. Daher wird in jedem Fall logarithmische Laufzeit für das sortierte Einfügen mittels binärer Suche benötigt.

Zum anderen stellt sich die Frage, in welche Baumknoten der neue Cluster $C = C_{i \cup j}$ eingefügt wird. Durch die Verwendung der Lance-Williams-Update-Formel wird die Wahl dieser Knoten stark eingeschränkt. Um die Distanz $D(C_{i \cup j}, C_p)$ ($C_p$ entspricht einem Singleton-Cluster mit dem Objekt $x_p$) durch die Lance-Williams-Update-Formel zu berechnen, benötigt man die Distanzen $D(C_i, C_p)$, $D(C_j, C_p)$ und $D(C_i, C_j)$. Die beiden Distanzen $D(C_i, C_p)$ und $D(C_j, C_p)$ existieren jedoch nur für die Knoten die sowohl $C_i$

als auch $C_j$ enthalten. Durch die in der Löschoperation beschriebene Abbildung aus Gleichung 5.1 können diese Knoten durch Finden des LCA (Lowest Common Ancestor) von $Blatt(C_i)$ und $Blatt(C_j)$ gefunden werden. Im folgenden Verlauf des Algorithmus gilt:

$$Blatt(C_{i\cup j}) = LCA(Blatt(C_i), Blatt(C_j)) \tag{5.2}$$

Durch dieses Vorgehen verringert sich die Baumtiefe und damit auch die Anzahl der verwendeten Pivots im Verlauf des Algorithmus. Es ist jedoch davon auszugehen, dass bevorzugt Cluster verschmolzen werden, welche im Pivot-Baum nahe beieinander oder im selben Knoten liegen. Eine Begründung hierfür ist, dass ein Cluster genau in dem Ast des Baumes liegt, zu dessen Pivot er den geringsten Abstand aufweist. Dadurch markieren die Pivots gewissermaßen Regionen im metrischen Raum. Die Tiefe des Baumes reduziert sich also erst, nachdem ein Großteil der lokalen Cluster verschmolzen wurde. Dadurch wird die Verteilung der Cluster im Raum immer spärlicher und die Distanzen müssen nicht mehr so exakt abgeschätzt werden.

Da der Baum durch die Einfügeoperation nicht tiefer wird, kann die Gesamtlaufzeit mit $\mathcal{O}(f\ \log_f(|P|)\ \log(n))$ abgeschätzt werden. Analog zur Analyse der Löschoperation wird $f$ im Folgenden als Konstante behandelt.

**Mögliche Verbesserung der Einfügeoperation** Es ist möglich die Verringerung der Baumtiefe durch die Einfügeoperation zu verhindern. Dazu muss jeder Pivot zusätzlich zu den Clustern in den Pivot-Baum eingefügt werden. Nach der Initialisierung der Datenstruktur mit Singleton-Clustern sind die Pivots also doppelt im Pivot-Baum vorhanden. Durch das Verschmelzen der Singleton-Pivot-Cluster werden die Pivot-Objekte jedoch nicht wieder aus dem Pivot-Baum gelöscht. Dadurch kann zu jedem Zeitpunkt die Distanz zwischen einem beliebigen Objekt und einem Pivot geschätzt werden. Verschmolzene Cluster könnten damit wie bei der Initialisierung des Pivot-Baumes immer dem Pivot zugeordnet werden, zu welchem sie die minimale Distanz besitzen. Dieses Verfahren sollte jedoch nur unterhalb des LCA angewandt werden, da durch die geschätzten Distanzen Folgefehler bei den Distanzabschätzungen entstehen können. Aufgrund dieser Folgefehler ist, ohne eine empirische Evaluation, keine fundierte Aussage darüber möglich, ob das Vorgehen einen Gewinn für die Clusteringqualität darstellt.

Die entsprechende Idee ist leider erst nach der Implementierung entstanden und konnte aus Zeitgründen auch nicht mehr implementiert werden. Daher bleibt die Untersuchungen der Effektivität einer späteren Analyse vorbehalten.

#### 5.3.2.2. Angepasstes Vorgehen bei der Best-Frontier-Suche

In der ursprünglichen Methode wird pro relevantem Baumknoten (Definition von relevant analog zur Löschoperation) eine eigene Suche nach dem nächsten Nachbarn von $C$ gestartet. Der ermittelte nächste Nachbar für jeden Baumknoten wird anschließend mit den

nächsten Nachbarn der anderen Knoten, auf Grundlage der exakten Distanzen, verglichen und derjenige mit geringster Distanz ausgewählt.

In dieser Arbeit wird ein unterschiedliches Vorgehen gewählt. Die Best-Frontier-Suche wird auf allen relevanten Distanzlisten in einem Durchlauf ausgeführt. Dadurch kann es vorkommen, dass einzelne Cluster durch die Best-Frontier-Suche unterschiedlich oft gezählt werden müssen, bevor die beste untere Schranke gefunden wurde. Dies ist darin begründet, dass nicht jeder Cluster ist allen Distanzlisten gespeichert ist. Gleichung 5.4 ist bereits aus Abschnitt 4.2.3.1 bekannt und verdeutlicht diesen Zusammenhang, denn wenn der nächste Nachbar von $C_i$ gesucht werden soll, dann unterschiedet sich die Kardinalität von $P_{i\cup j}$ für unterschiedliche $C_j$. Daher muss die Abbruchbedingung der Best-Frontier-Suche entsprechend angepasst werden.

$$P_{i\cup j} = \{C_p \in P_k \mid C_i, C_j \in X_k\} \tag{5.3}$$

$$\tilde{d}(C_i, C_j) = \max_{C_p \in P_{i\cup j}} |d(C_p, C_i) - d(C_p, C_j)| \tag{5.4}$$

Das Vorgehen der ursprünglichen Variante kommt für den Einsatzzweck in dieser Arbeit aus zwei Gründen nicht in Frage. Zum einen ist es in dem heuristischen SAHN-Clusterverfahren, nicht möglich die exakten Linkage-Distanzen effizient zu berechnen. Daher kann nicht bestimmt werden, welcher nächste Nachbar tatsächlich der Beste ist. Zum anderen produziert die modifizierte Variante, unabhängig von der Möglichkeit die exakten Distanzen zu berechnen, bessere Ergebnisse. Der Einfachheit halber wird im Folgenden für die $k$-Nächste-Nachbar-Suche mit $k = 1$ erläutert, worin das Problem der ursprünglichen Variante begründet ist. Die Erläuterung lässt sich jedoch auf jede beliebige Wahl von $k$ übertragen. Nehmen wir an, für jeden einzelnen Baumknoten $N_i$ wurde der nächste Nachbar von $x$ mit Hilfe der Pivots $P_i$ geschätzt. Es kann vorkommen, dass keiner dieser Kandidaten der echte nächste Nachbar von $x$ ist. In diesem Fall wäre es möglich, dass alle Kandidaten durch einen Pivot eines anderen Baumknotens verworfen worden wären und der echte nächste Nachbar gefunden würde. Daher gibt es Fälle, in denen durch die Modifikation der nächste Nachbar gefunden werden kann, in denen die originale Variante falsch gelegen hätte.

### 5.3.3. Performanzanalyse

Dieser Abschnitt analysiert die Laufzeit, den Speicherbedarf und die notwendige Anzahl an exakten Distanzberechnungen für das heuristische SAHN-Clusterverfahren.

#### 5.3.3.1. Laufzeitanalyse

Die Gesamtlaufzeit des heuristischen SAHN-Clusterverfahrens setzt sich aus der Laufzeit des modifizierten Generic-Clustering-Algorithmus und der Nächste-Nachbar-Suche mittels Best-Frontier-Verfahren zusammen.

Die Laufzeit in Bezug auf den modifizierten Generic-Clustering-Algorithmus ist bereits in Abschnitt 5.3.1 ausführlich diskutiert worden und beträgt $\mathcal{O}(n^2\ (\log(n) + k))$ bzw. $\Omega(n\ (\log(n) + k))$ wenn $k$ die Laufzeit der Nächste-Nachbar-Suche ist. Hinzu kommen die Update-Operationen (vgl. Abschnitt 5.3.2) des Pivot-Baumes für jeden verschmolzenen Cluster. Insgesamt werden exakt $n - 1$ Verschmelzungen vorgenommen, die jeweils eine Komplexität von $\mathcal{O}(\log(|P|)\ \log(n))$ besitzen (2 Löschoperationen + 1 Einfügeoperation). Die Gesamtlaufzeit ist daher $\mathcal{O}(n^2\ (\log(n) + k) + n\ \log(|P|)\ \log(n))$ bzw. $\Omega(n\ (\log(n) + k) + n\ \log(|P|)\ \log(n))$. Da $P \subseteq \mathcal{X} \Rightarrow |P| \leq n$ gilt, kann die Laufzeit verkürzt durch $\mathcal{O}(n^2\ (\log(n) + k))$ bzw. $\Omega(n\ (\log(|P|)\ \log(n) + k))$ ausgedrückt werden.

Während der Initialisierung des dynamischen Pivot-Baumes für die Best-Frontier-Suche werden pro Knoten $f = |P_i|$ (s. a. Abschnitt 4.2.3.1) verschiedene Pivots generiert. Für jede Split-Operation müssen alle Elemente dem nächsten Pivot zugeordnet werden. Bei konstanter Größe von $f$ liegt die Anzahl der Distanzberechnungen in diesem Fall bei $\mathcal{O}(|X_i|)$ für jeden Knoten $N_i$. Da jedes Objekt auf jeder Ebene des Pivot-Baumes genau einmal vorkommt, kann der Aufwand für jede Ebene des Pivot-Baumes mit $\mathcal{O}(n)$ abgeschätzt werden. Die gleiche Argumentation kann auf die Initialisierung der Distanzlisten $L_i$ angewandt werden. Pro Ebene des Pivot-Baumes liegt der Aufwand zur Initialisierung dieser Listen daher bei $\mathcal{O}(n\ \log(n))$. Die Gesamtlaufzeit der Initialisierung des dynamischen Pivot-Baumes für die Best-Frontier-Suche liegt damit bei $\mathcal{O}(\log(|P|)\ n\ \log(n))$.

Die Laufzeit der Best-Frontier-Suche wurde ebenfalls bereits in Abschnitt 4.2.3.2 diskutiert. Sie ist durch $\mathcal{O}(|P|\ \log(|P|)\ s)$[1] beschränkt. In Kombination mit dem Pivot-Baum ist die Anzahl der für eine Nächste-Nachbar-Suche genutzten Pivots durch $f\ \log(|P|)$ beschränkt. Für einen konstanten Wert $f$ beträgt die Laufzeit für eine Nächste-Nachbar-Suche in Kombination mit dem Pivot-Baum daher $\mathcal{O}(\log(|P|)\ \log(\log(|P|))\ s)$.

Durch Ersetzen von $k$ in der Laufzeit des modifizierten Generic-Clustering-Algorithmus erhält man schließlich eine Gesamtlaufzeit von $\mathcal{O}(n^2\ (\log(n) + \log(|P|)\ \log(\log(|P|))\ s))$ bzw. $\Omega(n\ (\log(|P|)\ \log(n) + \log(|P|)\ \log(\log(|P|))\ s))$. Wenn $s \in \Theta(\log(n))$ gewählt wird, dann lässt sich die Gesamtlaufzeit vereinfacht als $\mathcal{O}(n^2\ \log(|P|)\ \log(\log(|P|))\ \log(n))$ bzw. $\Omega(n\ \log(|P|)\ \log(\log(|P|))\ \log(n))$ ausdrücken.

#### 5.3.3.2. Speicherbedarf

Der Speicherbedarf des heuristischen SAHN-Clusterverfahrens wird durch die Distanzlisten der Pivots dominiert. Diese beinhalten konstant große Tupel von Objekten und Distanzen.

[1] Die Variable $s$ bezeichnet weiterhin die maximale Suchtiefe der Best-Frontier-Suche

Die Anzahl dieser Tupel entspricht der Kardinalität der Menge $X_i$ des entsprechenden Baumknotens $N_i$. Pro Baumebene summiert sich, analog zur Analyse der Laufzeit, die Anzahl der Tupel auf $f \cdot n$. In Kombination mit der Baumtiefe $\Theta(\log(P))$ und einer konstanten Wahl von $f$ ergibt sich daher ein Speicherbedarf von $\Theta(n \ \log(|P|))$.

Der Speicherbedarf des Pivot-Baumes kann mit dem Speicherbedarf der Distanzlisten verrechnet werden. Für jeden Knoten $N_i$ muss die Menge $X_i$, die Menge der Pivots $P_i$ und die Elternbeziehung gespeichert werden. Da $|P_i| \leq |X_i|$ gilt und die Kardinalität von $X_i$ mit der Anzahl der Tupel in den Distanzlisten übereinstimmt, liegt der Speicherbedarf für den Pivot-Baum ebenfalls in $\Theta(n \ \log(|P|))$.

Als weitere Datenstrukturen kommen die Hashing-Datenstruktur entsprechend der Gleichung 5.1 und die Prioritätswarteschlangen des Generic-Clustering-Algorithmus und der Best-Frontier-Suche zum Einsatz. Die Hashing-Datenstruktur und die Prioritätswarteschlange des Generic-Clustering-Algorithmus speichern $\mathcal{O}(n)$ und die Prioritätswarteschlange der Best-Frontier-Suche $\mathcal{O}(\log(|P|))$ Elemente (maximal zwei pro verwendetem Pivot). Der asymptotische Gesamtspeicherbedarf wird durch diese Datenstrukturen daher nicht beeinflusst und liegt in $\Theta(n \ \log(|P|))$.

#### 5.3.3.3. Anzahl exakter Distanzberechnungen

Die Anzahl der exakten Distanzberechnungen pro Distanzliste ist die Größe der Distanzliste selbst, da für jedes Objekt die Distanz zu dem entsprechenden Pivot berechnet werden muss. Analog zur Analyse der Laufzeitzeit, ist erneut für jedes Baumlevel die Anzahl der Distanzberechnungen $f \cdot n$. Für ein konstantes $f$ ist die Gesamtanzahl der exakten Distanzberechnungen damit $\Theta(n \ \log(|P|))$.

## 5.4. Implementierung

Dieser Abschnitt befasst sich mit der Implementierung des heuristischen SAHN-Clusterverfahrens und der Einbettung in bestehende Programmstrukturen. Es wird ausschließlich auf die Modellschicht des Model-View-Controller (MVC) Entwurfsmusters (s. a. [Mö98]) eingegangen. Da es verschiedene Auslegungen des MVC-Entwurfsmusters gibt, sei an dieser Stelle festgehalten, dass die Geschäftslogik Teil des Modells ist.

### 5.4.1. Bestehende Programmstruktur

Die bisherige Programmstruktur ist geprägt durch die Tatsache, dass nur ein bestimmtes Clusterverfahren implementiert werden sollte. Zum Zeitpunkt der Implementierung war noch nicht abzusehen, das weitere hierarchische Clusterverfahren implementiert werden sollen, bzw. welche Anforderungen diese an die Programmstruktur haben. Das stark vereinfachte Klassendiagramm ist in Abbildung 5.1 dargestellt.

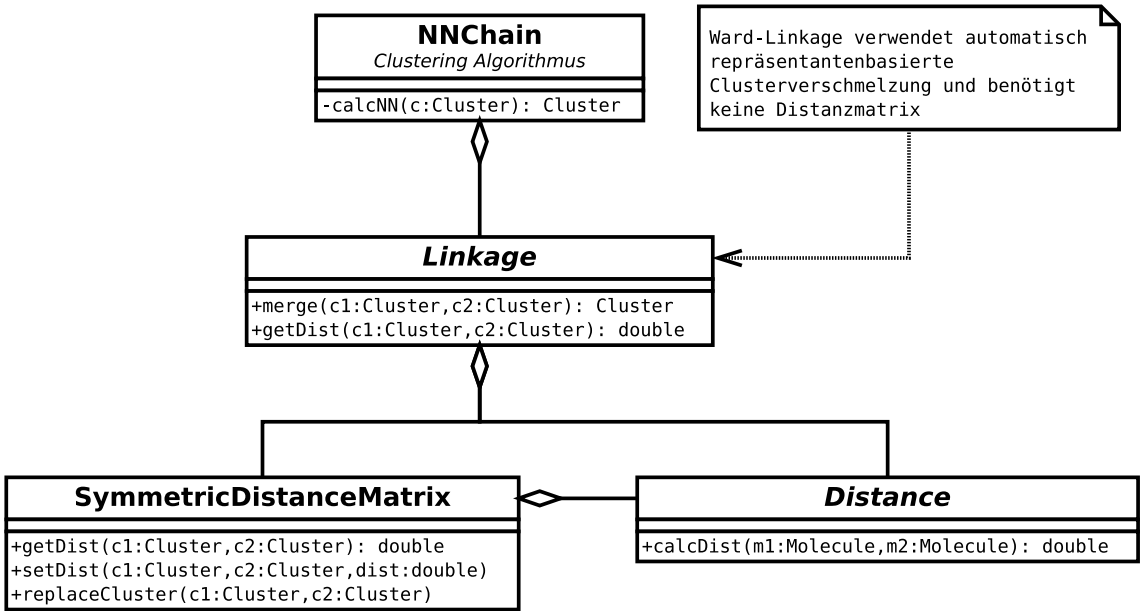



**Abbildung 5.1.:** Altes Clustering-Modell (Scaffold Hunter 2.0)

Die Klasse *NNChain* implementiert den NNChain-Algorithmus und die Nächste-Nachbar-Suche. Sie erhält die Inter-Cluster-Distanzen von der Klasse *Linkage*, die als Proxy zur Klasse *SymmetricDistanceMatrix* fungiert. Da die Distanzupdates vom konkreten Linkage-Verfahren abhängig sind, wird die Matrix nicht direkt in der Klasse *NNChain* aggregiert. Ein Großteil der Programmlogik bzgl. der Verschmelzung von Clustern ist daher in der *Linkage*-Klasse definiert.

Die Klasse *Linkage* entscheidet zusätzlich darüber, ob eine Distanzmatrix benötigt wird. Für Ward-Linkage kann in Zusammenhang mit dem euklidischen Distanzmaß eine NNChain-Version mit Repräsentanten verwendet werden. Dies führt zu einem linearen Speicherbedarf und einer etwas schlechteren Laufzeit (vgl. Abschnitt 4.1.3.1). Es wurde zum Zeitpunkt der Modellierung festgelegt, die Variante mit linearem Speicherplatz grundsätzlich der Variante mit quadratischem Speicherplatz vorzuziehen. Daher wurde nicht vorgesehen, dieses Verhalten zu beeinflussen.

Die Klasse *Distance* wird zum einen zur Initialisierung der Distanzmatrix benötigt und zum anderen zur Berechnung der Distanz zwischen zwei Clustern, falls die repräsentantenbasierte NNChain-Variante verwendet wird. Daher ist *Distance* sowohl ein Teil von *Linkage* als auch von *SymmetricDistanceMatrix*.

### 5.4.2. Modularisierung der bestehenden Programmstruktur

In Bezug auf diese Arbeit und der damit einhergehenden Implementierung unterschiedlicher Clusterverfahren und Nächste-Nachbar-Suchstrategien ist die zugrundeliegende Programmstruktur nicht flexibel genug. Insbesondere für die Evaluation der unterschiedlichen

Verfahren ist es notwendig, verschiedene Module (möglichst) frei kombinieren zu können, um deren Einfluss auf die Clusteringqualität und Performanz bewerten zu können.

Sollen unterschiedliche Nächste-Nachbar-Suchstrategien zum Einsatz kommen, können diese nicht wie bisher im Clustering implementiert werden. Die Art und Weise der Nächste-Nachbar-Suche ist stark von der zugrundeliegenden Datenstruktur abhängig. Deshalb ist es wünschenswert die Nächste-Nachbar-Suchstrategie enger mit ihr zu vereinen. Würde die *Linkage*-Klasse weiterhin als Proxy zwischen dieser Datenstruktur und der Nächste-Nachbar-Suche fungieren, müsste sie eine Vielzahl unterschiedlicher Anfragen beantworten können. Die Semantik der Klasse *Linkage* würde zudem weit gedehnt, da viele Programmteile, die logisch nicht mit dem Linkage-Verfahren verknüpft sind, von dieser Klasse umgesetzt werden müssten. Daher wurde die Klassenstruktur gemäß Abbildung 5.2 angepasst.

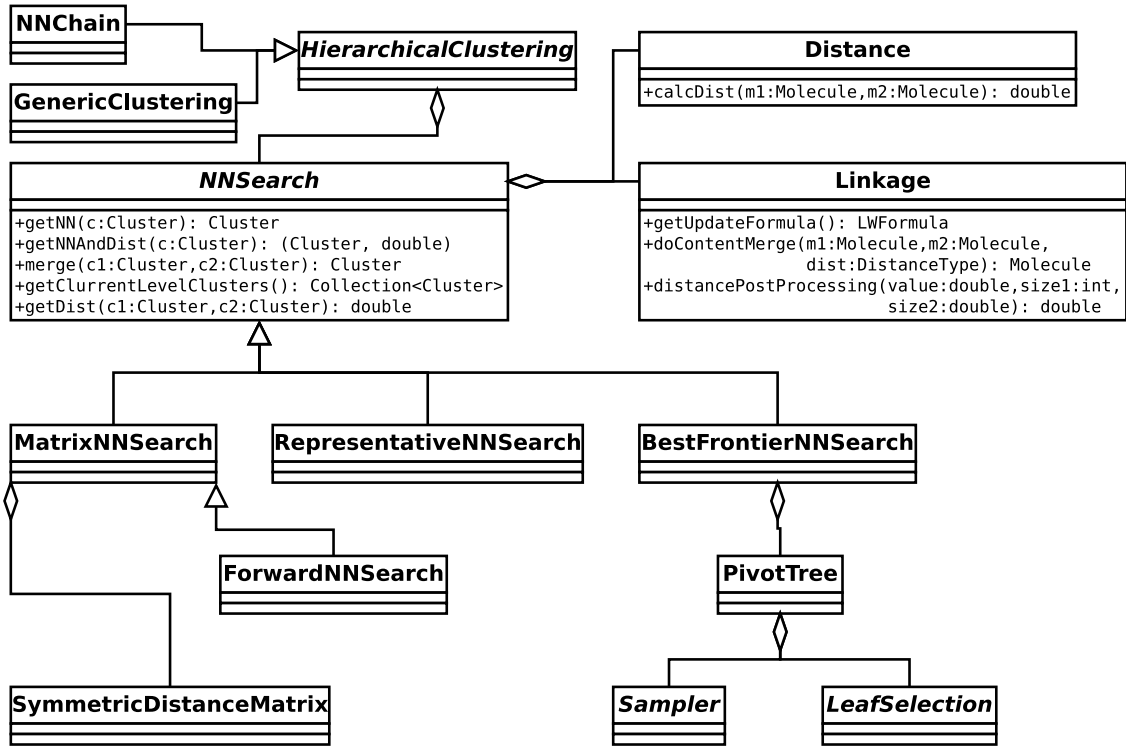



**Abbildung 5.2.:** Neues Clustering-Modell

Der Clustering-Algorithmus (Interface: *HierarchicalClustering*) wurde von der Nächste-Nachbar-Suche getrennt. Daher gibt es ein neues Interface *NNSearch*, dessen abgeleitete Klassen unterschiedliche Nächste-Nachbar-Suchstrategien implementieren. Jede Nächste-Nachbar-Suchstrategie verwendet intern eine passende Datenstruktur. Die Klasse *MatrixNNSearch* stellt die bisher implementierte exakte Nächste-Nachbar-Suche auf der Distanzmatrix dar und die Klasse *ForwardNNSearch* implementiert die Vorwärtssuche für den Generic-Clustering-Algorithmus (vgl. Abschnitt 4.1.3.3). Da sie ebenfalls eine Distanzmatrix benötigt und sich nur marginal von der Klasse *MatrixNNSearch* unterscheidet, ist sie von ihr abgeleitet. Die Klasse *RepresentativeNNSearch* stellt die Variante für die

Nächste-Nachbar-Suche mittels Repräsentanten dar. Diese Nächste-Nachbar-Suchstrategie wurde bisher für Ward-Linkage und das euklidischen Distanzmaß verwendet, kann jedoch auch für Median- und Zentroid-Linkage in Kombination mit dem Generic-Clustering-Algorithmus eingesetzt werden. Die Klasse *BestFrontierNNSearch* implementiert die Best-Frontier-Suche mit dem dynamischen Pivot-Baum als Datenstruktur.

Das Interface *Linkage* wurde auf die Funktionen reduziert, welche semantisch direkt mit dem Linkage-Verfahren zusammenhängen. Die Implementierungen können eine Lance-Williams-Update-Formel zurückgeben, mit dessen Hilfe unterschiedliche Nächste-Nachbar-Suchstrategien die Distanzupdates berechnen können. Durch diese Trennung von *Linkage* und *LanceWilliamsUpdateFormula* ist es möglich das Interface *Linkage* relativ schlank zu halten und die Programmlogik in den Nächste-Nachbar-Suchstrategien zu implementieren. Für repräsentantenbasierte Nächste-Nachbar-Suchstrategien ist die Lance-Williams-Update-Formel jedoch nicht ausreichend. Daher implementieren die *Linkage*-Klassen eine Methode *doContentMerge* mit der ein neuer Repräsentant auf Grundlage der zu verschmelzenden Repräsentanten und dem Distanzmaß berechnet werden kann.

Der Übersicht halber wurden in Abbildung 5.2 nur die wichtigsten Teile aufgeführt, weswegen im Folgenden einige noch fehlende Konzepte, Klassen und Methoden vorgestellt werden.

Jedes Modul (*HierarchicalClustering*, *Distance*, *NNSearch*, *Linkage*) verfügt über Funktionen, die eine vollautomatische Kompatibilitätsprüfung der einzelnen Module ermöglichen. Die konkreten Implementierungen des Interfaces *Linkage* können z. B. Auskunft darüber geben, ob sie die Reducibility-Eigenschaft erfüllen oder nicht. Durch dieses Vorgehen kann die Benutzeroberfläche dynamisch die passenden Linkage-Verfahren, Distanzmaße, etc. anzeigen, ohne bei jeder Änderung von Modulen ebenfalls angepasst werden zu müssen.

Um die Erstellung der einzelnen Module möglichst einfach zu gestalten, wurde das Factory-Entwurfsmuster verwendet. Für jede Modulklasse existiert ein *Enum*, welches die entsprechende Factory-Methode enthält. Des weiteren sind die im letzten Absatz beschriebenen Kompatibilitätsprüfmethoden direkt über das *Enum* zugreifbar und daher müssen die verschiedenen Klassen nicht instanziiert werden, um statische Informationen abzufragen. Der Vorteil gegenüber *static* Methoden (die den gleichen Zweck erfüllen könnten) liegt darin begründet, dass *Enum*-Werte einfach als Parameter weitergereicht und alle verfügbaren Module aufgelistet werden können. Dieses Vorgehen erlaubt eine strikte Trennung der Benutzeroberfläche von dem Modell, da kein Vorwissen über existierende Module und Klassen in der Benutzeroberfläche vorhanden sein muss.

### 5.4.3. Pivot-Baum-Datenstruktur

Da die Pivot-Baum-Datenstruktur einen zentralen Teil der Best-Frontier-Suche darstellt, ist sie in Abbildung 5.3 detaillierter dargestellt.

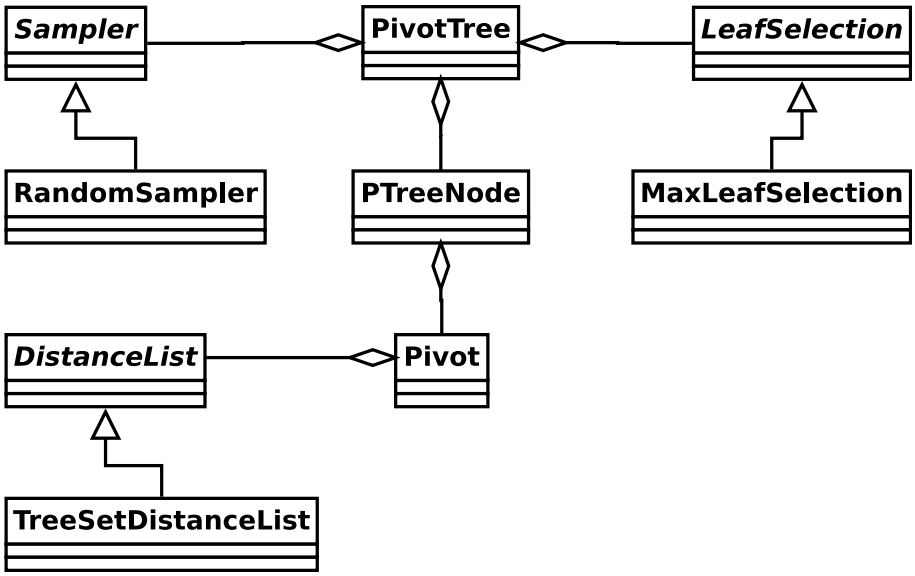



**Abbildung 5.3.:** Pivot-Baum-Datenstruktur

Der Pivot-Baum besteht aus einer Menge von *PTreeNode*-Objekten. Jeder *PTreeNode* kennt seinen Elter-Knoten. Die Klasse *PivotTree* enthält zum Finden desjenigen Blattes, welches einen bestimmen Cluster enthält, eine *HashMap* (vgl. Gleichung 5.1). Über das Blatt und die Elter-Beziehung kann, wie in Abschnitt 5.3.2 beschrieben, die Menge der relevanten Pivots für die Nächste-Nachbar-Suche, die Löschoperation und die Einfügeoperation ermittelt werden.

Jeder *PTreeNode* enthält wiederum eine Menge von *Pivot*-Objekten und diese enthalten eine Klasse des Typs *DistanceList*. Diese Distanzliste ist sortiert nach der Distanz zum Pivot und muss das schnelle Auffinden eines Objekts ermöglichen. Um die Best-Frontier-Suche effizient zu unterstützen, muss die Liste, von einem gefundenen Objekt ausgehend, in beide Richtungen traversiert werden können. Umgesetzt wurde das Interface *DistanceList* in der Klasse *TreeSetDistanceList*. Wie der Name vermuten lässt, wird ein Java *TreeSet*[2] verwendet, welches wiederum eine Rot-Schwarz-Baum-Implementierung darstellt. Mit Hilfe dieser Datenstruktur kann ein Objekt in $\mathcal{O}(\log(n))$ gefunden und sortiert eingefügt werden. Die Traversierung der Liste ist in amortisiert $\mathcal{O}(n)$ möglich. Diese Laufzeiten sind zum Teil nicht optimal, denn durch Verwendung einer Hashing-Datenstruktur kann das Auffinden eines Elementes in konstanter Zeit realisiert werden. Auch die Traversierung kann in nicht-amortisiert $\mathcal{O}(n)$ realisiert werden. Da für diesen Zweck jedoch keine passende Datenstruktur in den Java Standardbibliotheken verfügbar war, wurde aus Zeitgründen auf eine

[2] http://docs.oracle.com/javase/6/docs/api/java/util/TreeSet.html

Eingenimplementierung verzichtet. Die Gesamtlaufzeit des Algorithmus verschlechtert sich daher um den Faktor $\log(n)$. In Abschnitt 6.5 wird mit Hilfe eines Profilers eine genaue Analyse der Laufzeitauswirkungen dieses Vorgehens untersucht.

Während der Initialisierung der Datenstruktur muss eine Menge von Pivots aus einer Menge von Objekten gezogen werden (vgl. Abschnitt 4.2.3.1). In der zugrundeliegenden Literatur wurde ein zufälliges Sampling verwendet. Es ist jedoch denkbar, alternative Sampling-Strategien (vgl. Abschnitt 4.2.1.1) zu verwenden oder verschiedene Samples auf ihre Effektivität als Pivot-Menge (vgl. Gleichung 4.17) zu untersuchen und das beste Sample zu verwenden. Daher wurde das Interface *Sampler* definiert und die Modellierung an dieser Stelle offen für alternative Strategien gehalten. In der Umsetzung dieser Arbeit wird das Interface *Sampler* ausschließlich von der Klasse *RandomSampler* implementiert. Die Klasse verwendet den Algorithmus von Floyd [BF87]. Er benötigt für die Ziehung eines $m$ großen Samples $\mathcal{O}(m)$ Rechenschritte.

In Abschnitt 5.3.2.1 wurde bereits erwähnt, dass unterschiedliche Strategien für die Auswahl des Blattes möglich sind, welches für die Split-Operation verwendet wird. *LeafSelection* stellt ein Inferface für eine solche Strategie dar. In dieser Arbeit wurde lediglich die Klasse *MaxLeafSelection* implementiert, welche den Split immer auf dem Blatt mit den meisten Objekten ausführt.

# Kapitel 6.

# Experimentelle Evaluation

Dieses Kapitel beschreibt die experimentelle Evaluation des heuristischen SAHN-Clusterverfahrens. Um die Laufzeit und Qualität des Verfahrens beurteilen zu können, wird ein Vergleich mit den exakten Algorithmen durchgeführt. In Abschnitt 6.1 wird daher zunächst ein kurzer Überblick über bekannte Qualitätsmaße zum Vergleich zweier Clusterings gegeben. Abschnitt 6.2 beschreibt die Implementierung eines Evaluationsframeworks in der Software Scaffold Hunter und Abschnitt 6.5 stellt die Ergebnisse dar.

## 6.1. Qualitätsmaße

Dieser Abschnitt beschäftigt sich mit der Frage, wie die Qualität eines Clusterings gemessen werden kann. Es werden geeignete Methoden vorgestellt und ausgewählt, welche in Abschnitt 6.5 verwendet werden.

### 6.1.1. Grundlegendes

Zur Bewertung der Qualität eines Clustering kommen zwei unterschiedliche Methoden in Frage.

Die erste Methode besteht in der Definition eines Qualitätskriteriums, welches gewisse Eigenschaften eines Clustering, unabhängig vom verwendeten Clusterverfahren, bewertet. So kann z. B. die Separation der Cluster, die Kompaktheit der Cluster oder die Form der Cluster als Maßstab dafür gelten, wie gut ein Clustering zu bewerten ist. Es ist ebenfalls möglich einen synthetischen Datensatz zu generieren, bei dem eine Grundwahrheit bekannt ist: D. h. bei dem die Cluster bereits bekannt sind. Anschließend wird untersucht, ob die generierten Cluster durch das Clusterverfahren erkannt werden. Dieses Vorgehen eignet sich gut, um neue Clusterverfahren zu testen und Klassen von Datensätzen zu finden, für die ein bestimmtes Clusterverfahren geeignet ist.

Für die Untersuchung des heuristischen SAHN-Clusterverfahrens eignen sich die soeben vorgestellten Qualitätskriterien jedoch nicht. Das exakte SAHN-Clusterverfahren zählt zu einem der bekanntesten Clusterverfahren und ist daher hinreichend auf seine Eignung für bestimmte Datensätze und die Qualität der Ergebnisse untersucht worden. Vielmehr muss

die Abweichung zwischen heuristischem und exaktem Verfahren untersucht werden, so dass die entscheidende Frage lautet: Wie groß ist der durch die Heuristik entstandene Fehler? Gesucht ist folglich ein Ähnlichkeits- oder Distanzmaß $Q(\mathcal{C}_1, \mathcal{C}_2)$ zwischen zwei Clusterings $\mathcal{C}_1$ und $\mathcal{C}_2$. Ein solches stellt die zweite Methode zur Untersuchung der Clusteringqualität dar. Sie wird im Folgenden verwendet, um die Qualität des heuristischen SAHN-Clusterverfahrens zu analysieren.

### 6.1.2. Spezialfall hierarchische Clusterverfahren

Die meisten Verfahren zum Vergleich von Clusterings setzen ein flaches (nicht hierarchisches) Clustering als Eingabe voraus. Da jedoch ein hierarchisches Clusterverfahren, durch Angabe eines Schwellenwertes, leicht in ein flaches Clustering transformiert werden kann, stellt diese Einschränkung keine Hürde bzgl. der Anwendbarkeit dar. Sei im Folgenden das Level $L_{k,i}$ der horizontale Schnitt im Dendrogramm des Clustering $\mathcal{C}_k$, bei dem $i$ unterschiedliche Cluster entstehen. Jedes Level ist somit eines der möglichen flachen Clusterings, die aus einem hierarchischen Clustering gewonnen werden können. $Q(L_{1,i}, L_{2,i})$ stellt das Qualitätsmaß zwischen den beiden hierarchischen Clusterings $\mathcal{C}_1$ und $\mathcal{C}_2$ auf Level $i$ dar.

Alle im Folgenden vorgestellten Verfahren zum Vergleich zweier hierarchischer Clusterings verwenden das folgende Schema: Für jedes Level $i$ wird $Q(L_{1,i}, L_{2,i})$ berechnet. Anschließend werden die Tupel $(i, Q(L_{1,i}, L_{2,i}))$ mit $1 \leq i \leq n$ in einem Plot dargestellt. Dadurch kann für jedes Level die Qualität des Clustering abgelesen und interpretiert werden.

### 6.1.3. Auswahl konkreter Qualitätsmaße

Im Folgenden ist eine Auswahl von Qualitätsmaßen aufgeführt. In der in Abschnitt 6.5 durchgeführten Evaluation wird der *Fowlkes-Mallows-Index* [FM83] sowie ein informationstheoretisches Maß mit dem Namen *Variation of Information* (VI) [Mei07] verwendet. Der *Rand-Index* [Ran71] ist der Vollständigkeit halber ebenfalls aufgeführt, da er eines der bekanntesten Qualitätsmaße darstellt. Weiterhin führte er grundlegende Techniken ein, welche von vielen anderen Qualitätsmaßen (u. a. Fowlkes-Mallows-Index) ebenfalls verwendet werden.

#### 6.1.3.1. Rand-Index

Der Rand-Index ist über die Kontingenzmatrix $M$ definiert:

**6.1.1 Definition (Kontingenzmatrix).** Die Kontingenzmatrix beinhaltet die Kardinalität aller Schnitte zwischen den Clustern zweier flachen Clusterings $\mathcal{C}_1$ und $\mathcal{C}_2$

$$M = [m_{ij}] \text{ mit } m_{ij} = |C_i \cap C_j|;\ C_i \in \mathcal{C}_1;\ C_j \in \mathcal{C}_2 \tag{6.1}$$

**6.1.2 Definition (Rand-Index).**

$$Q_R(\mathcal{C}_1, \mathcal{C}_2) = \frac{\binom{n}{2} - \frac{1}{2}\left(\sum_i(\sum_j m_{ij})^2 + \sum_j(\sum_i m_{ij})^2\right) - \sum_i \sum_j m_{ij}^2}{\binom{n}{2}} \tag{6.2}$$

Der Rand-Index stellt ein 1-genormtes Ähnlichkeitsmaß dar. Existiert kein Objektpaar $(x_i, x_j)$, so dass gilt $x_i, x_j \in C_1 \wedge x_i, x_j \in C_2 \wedge C_1 \in \mathcal{C}_1 \wedge C_2 \in \mathcal{C}_2$, dann nimmt $Q_R$ den Wert 0 an. Sind die Clusterings identisch ist der Wert $Q_R = 1$.

Eine alternative Art und Weise den Rand-Index zu definieren, besteht darin die Paare von Objekten zu zählen, die sowohl in $\mathcal{C}_1$ als auch in $\mathcal{C}_2$ im selben Cluster oder verschiedenen Clustern liegen. Deshalb zählt der Rand-Index zur Klasse der *Counting-Pairs*-Qualitätsmaße (s. a. [Mei07]). Es ist daher möglich den Rand-Index mit Hilfe von Bitvektoren zu definieren. Der Bitvektor $B_1$ des Clusterings $\mathcal{C}_1$ besitzt für jedes Objektpaar ein Bit. Liegt das Objektpaar in einem gemeinsamen Cluster, so wird das entsprechende Bit auf 1 gesetzt. Im Folgenden bezeichnet $N_{11}$ die Menge der gemeinsamen 1 Bits und $N_{00}$ die Menge der gemeinsamen 0 Bits der Bitvektoren $B_1$ und $B_2$. Diese Notation ist bereits aus Abschnitt 2.1.2.1 (Tanimoto Distanzmaß) bekannt.

**6.1.3 Definition (Rand-Index – Bitvektor Variante).**

$$Q_R(\mathcal{C}_1, \mathcal{C}_2) = \frac{|N_{11}| + |N_{00}|}{\binom{n}{2}} \tag{6.3}$$

Ein Problem in Bezug auf den Rand-Index besteht darin, dass der Erwartungswert $E[Q_R(\mathcal{C}_1, \mathcal{C}_2)]$ für zwei zufällig bestimmte, unabhängige Clusterings abhängig von der Clusteranzahl $k = |\mathcal{C}_1|$ und $l = |\mathcal{C}_2|$ ist. Für den Vergleich hierarchischer Clusterings kann im Folgenden $k = l$ angenommen werden, da immer $L_{1,i}$ mit $L_{2,i}$ verglichen wird. Für $k = 2$ liegt der Erwartungswert $E[Q_R(\mathcal{C}_1, \mathcal{C}_2)]$ bei etwa 0,5 und für $k > 2$ nähert er sich mit zunehmender Größe immer weiter dem Wert 1 an (vgl. Abbildung 6.1). Dieser Zusammenhang ist besonders ungünstig, da der relevante Wertebereich immer kleiner wird, je größer $k$ gewählt wird. Schon bei $k = 50$ liegt der Wertebereich, in dem ein Zusammenhang zwischen den Clusterings $\mathcal{C}_1$ und $\mathcal{C}_2$ angenommen werden kann, zwischen 0,95 und 1.

Um dieses Problem zu umgehen, wurde der normalisierte Rand-Index eingeführt:

**6.1.4 Definition (Normalisierter Rand-Index).**

$$Q_{NR}(\mathcal{C}_1, \mathcal{C}_2) = \frac{Q_R(\mathcal{C}_1, \mathcal{C}_2) - E[Q_R(\mathcal{C}_1, \mathcal{C}_2)]}{1 - E[Q_R(\mathcal{C}_1, \mathcal{C}_2)]} \tag{6.4}$$

Für den Rand-Index gilt aufgrund des nicht konstanten Erwartungswerts, dass ein Wert $y$ für $k = x$ nicht vergleichbar mit einem Wert $z$ für $k \neq x$ ist. Für das hierarchische Clustering stellt dies ein Problem dar, da auch der Plot der Werte $(i, Q(L_{1,i}, L_{2,i}))$ verzerrt wird. Ein vermeintlicher Anstieg der Clusteringqualität mit zunehmender Clusteranzahl kann alleine auf die Eigenschaften des Qualitätsmaßes zurückzuführen sein.

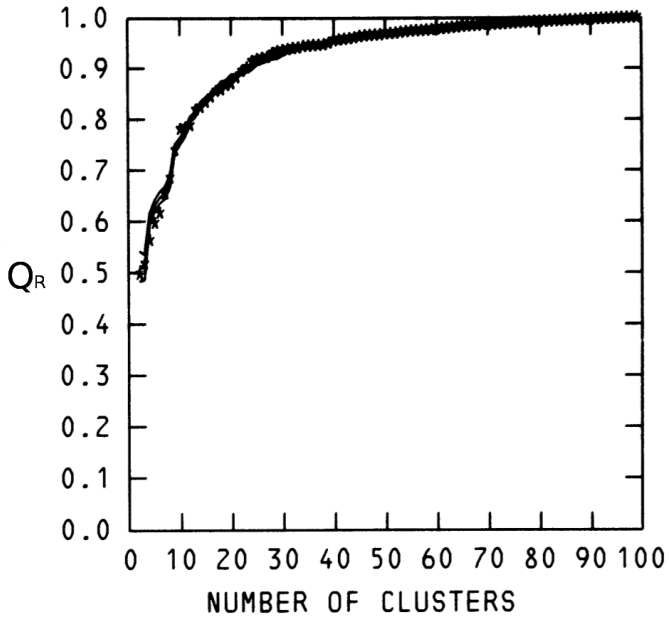



**Abbildung 6.1.:** $Q_R(\mathcal{C}_1, \mathcal{C}_2)$ für zwei unabhängige Clusterings, Quelle: [FM83]

Diese Einschränkung bzgl. der Vergleichbarkeit gilt auch für den normalisierten Rand-Index, da keine Linearität des Wertebereichs für ein beliebiges aber festes $k$ angenommen werden kann. Es ist also ebenfalls unklar, wie verschiedene Werte für solches $k$ miteinander zu vergleichen sind. Der Wertebereich kann nur als Ordinalskala interpretiert werden. Diese Erkenntnis hat weitreichende Folgen für die Interpretierbarkeit der Werte. Wenn z. B. $Q_{NR}(\mathcal{C}_1, \mathcal{C}_2)$ und $Q_{NR}(\mathcal{C}_2, \mathcal{C}_3)$ bekannt sind, lässt sich daraus keine Aussage über den Wert $Q_{NR}(\mathcal{C}_1, \mathcal{C}_3)$ ableiten.

#### 6.1.3.2. Fowlkes-Mallows-Index

Der Fowlkes-Mallows-Index (s. a. [FM83]) gehört, genauso wie der Rand-Index, zur Klasse der *Counting-Pairs*-Qualitätsmaße. Wie alle Qualitätsmaße dieser Klasse kann er über die Kontingenzmatrix $M$ definiert werden:

**6.1.5 Definition (Fowlkes-Mallows-Index).**

$$Q_{FM}(\mathcal{C}_1, \mathcal{C}_2) = \frac{\sum_i \sum_j m_{ij}^2 - n}{\sqrt{P\ R}} \tag{6.5}$$

$$m_{\cdot j} = \sum_i m_{ij} \tag{6.6}$$

$$m_{i\cdot} = \sum_j m_{ij} \tag{6.7}$$

$$P = \sum_i (m_{i\cdot})^2 - n \tag{6.8}$$

$$R = \sum_j (m_{\cdot j})^2 - n \tag{6.9}$$

Der Fowlkes-Mallows-Index ist ebenfalls ein 1-normiertes Ähnlichkeitsmaß. Er unterscheidet sich vom Rand-Index jedoch im Erwartungswert zweier unabhängiger Clusterings bzgl. $k$ (vgl. Abbildung 6.2).

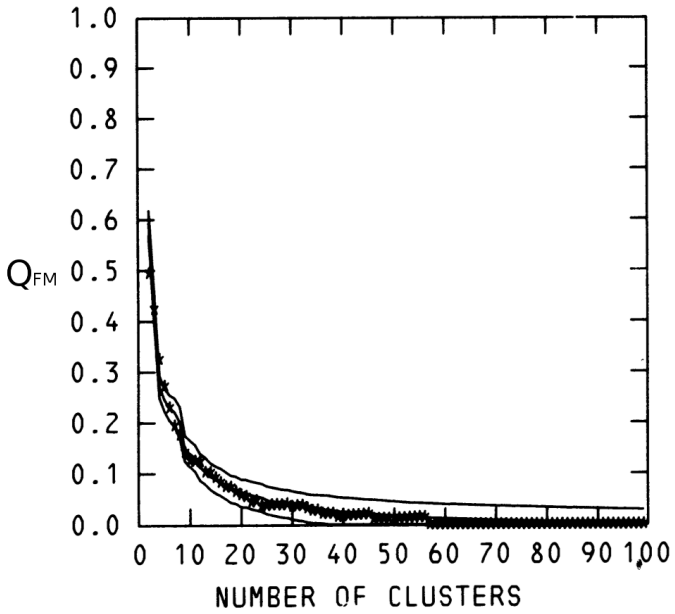



**Abbildung 6.2.:** $Q_{FM}(\mathcal{C}_1, \mathcal{C}_2)$ für zwei unabhängige Clusterings, Quelle: [FM83]

Die in Abschnitt 6.1.3.1 beschriebenen Probleme bzgl. der Vergleichbarkeit von Werten für unterschiedliche $k$ und der Nichtlinearität des Wertebereichs gelten auch für den Fowlkes-Mallows-Index. Ein Vorteil des Fowlkes-Mallows-Index besteht jedoch darin, dass die Autoren den Erwartungswert (Gleichung 6.10) und die Varianz (Gleichung 6.11) benennen. Daher ist das Ergebnis trotz nicht konstantem Erwartungswert relativ gut interpretierbar. Es kann ein Schwellenwert festgelegt werden, ab dem ein Zusammenhang zwischen zwei Clusterings erwartet wird. Gegenüber dem Rand-Index wird zudem der relevante Wertebereich für große $k$ nicht verkleinert. Daher wird in dieser Arbeit der

Fowlkes-Mallows-Index dem Rand-Index vorgezogen.

#### Erwartungswert für unabhängige Clusterings

$$E[Q_{FM}(\mathcal{C}_1, \mathcal{C}_2)] = \frac{\sqrt{P\ R}}{n(n-1)} \tag{6.10}$$

#### Varianz für unabhängige Clusterings

$$\begin{aligned} \sigma^2[Q_{FM}(\mathcal{C}_1, \mathcal{C}_2)] = & \frac{2}{n(n-1)} + \frac{4\ P'\ R'}{n(n-1)(n-2)\ P\ R} \\ & + \frac{P - 2 - 4\ \frac{P'}{P})(R - 2 - 4\ \frac{R'}{R})}{n(n-1)(n-2)(n-3)} - \frac{P\ R}{n^2(n-1)^2} \end{aligned} \tag{6.11}$$

$$P' = \sum_i m_{i\cdot}\ (m_{i\cdot} - 1)(m_{i\cdot} - 2) \tag{6.12}$$

$$R' = \sum_j m_{\cdot j}\ (m_{\cdot j} - 1)(m_{\cdot j} - 2) \tag{6.13}$$

Die Formel für die Varianz ist schwer interpretierbar. Daher ist in Abbildung 6.2 die zweifache Standardabweichung vom Erwartungswert eingezeichnet (schwarze Linien oberhalb und unterhalb der Kurve).

Wie bei dem Rand-Index existiert eine normierte Variante des Fowlkes-Mallows-Index. Da für diese Variante jedoch die gleichen Einschränkungen wie bei dem Rand-Index gelten (nicht linearer Wertebereich), ist durch sie keine bessere Interpretierbarkeit gegeben. Für die Interpretation des nicht-normierten Fowlkes-Mallows-Index muss der Erwartungswert immer als Referenz herangezogen werden, die Verzerrung des Qualitätsmaßes wird jedoch nicht durch die Normierung verschleiert. Daher wird im weiteren Verlauf dieser Arbeit der nicht-normierte Fowlkes-Mallows-Index verwendet.

### 6.1.3.3. Variation of Information

Im Gegensatz zum Rand-Index oder dem Fowlkes-Mallows-Index beruht das Qualitätsmaß *Variation of Information* (VI) [Mei07] nicht auf dem Vergleich der Clusterzugehörigkeit von Objektpaaren, sondern auf der Entropie der Clusterings. VI behebt teilweise die Probleme der anderen vorgestellten Qualitätsmaße und ist daher besser interpretierbar.

Das Qualitätsmaß VI ist über die Entropie der Clustergrößen-Verteilung definiert. Die Wahrscheinlichkeit, dass ein beliebiges Objekt Element eines Clusters $C$ ist, ist definiert als:

$$P(C) = \frac{|C|}{n} \tag{6.14}$$

Die Entropie der Clustergrößen-Verteilung ist daher folgendermaßen zu berechnen:

$$H(\mathcal{C}) = -\sum_{C \in \mathcal{C}} P(C)\ \log(P(C)) \tag{6.15}$$

Die Entropie eines einzelnen Clusterings hilft jedoch noch nicht zwei Clusterings miteinander zu vergleichen. Um diesen Zusammenhang zu modellieren, wird die Transinformation $I(\mathcal{C}_1, \mathcal{C}_2)$ (symmetrisch) der beiden Clustergrößen-Verteilungen verwendet. Die Transinformation gibt an, um wie viel die Entropie $H(\mathcal{C}_1)$ reduziert wird, wenn $\mathcal{C}_2$ bereits bekannt ist.

$$I(\mathcal{C}_1, \mathcal{C}_2) = \sum_{C_1 \in \mathcal{C}_1} \sum_{C_2 \in \mathcal{C}_2} P(C_1, C_2) \ \log \left( \frac{P(C_1, C_2)}{P(C_1)\ P(C_2)} \right) \tag{6.16}$$

wobei:

$$P(C_1, C_2) = \frac{|C_1 \cap C_2|}{n} \tag{6.17}$$

Das Distanzmaß $Q_{VI}$ lässt sich dann folgendermaßen definieren:

**6.1.6 Definition (VI ($Q_{VI}$)).**

$$Q_{VI}(\mathcal{C}_1, \mathcal{C}_2) = H(\mathcal{C}_1) + H(\mathcal{C}_2) - 2I(\mathcal{C}_1, \mathcal{C}_2) \tag{6.18}$$

Intuitiv ausgedrückt bezeichnet $Q_{VI}$ damit den Anteil der Entropie der beiden Clusterings, welcher nicht durch die Kenntnis des jeweils anderen Clusterings beseitigt werden kann: $Q_{VI}(\mathcal{C}_1, \mathcal{C}_2) = H(\mathcal{C}_1|\mathcal{C}_2) + H(\mathcal{C}_2|\mathcal{C}_1)$. Im Falle identischer Clusterings gilt: $\mathcal{C}_1 = \mathcal{C}_2 \Rightarrow H(\mathcal{C}_1) = H(\mathcal{C}_2) = I(\mathcal{C}_1, \mathcal{C}_2)$. Daher nimmt $Q_{VI}$, für identische Clusterings, den Wert 0 an. Der Wertebereich von $I(\mathcal{C}_1, \mathcal{C}_2)$ liegt zwischen 0 und $\min(H(\mathcal{C}_1), H(\mathcal{C}_2))$ und für $Q_{VI}$ gilt $0 \leq Q_{VI} \leq log(n)$.

In der Veröffentlichung [Mei07] wird bewiesen, dass es sich bei $Q_{VI}$ um eine Metrik handelt. Durch die Dreiecksungleichung kann daher eine gewisse Transitivität angenommen werden: Sind sich beispielsweise $\mathcal{C}_1$ und $\mathcal{C}_2$ sowie $\mathcal{C}_2$ und $\mathcal{C}_3$ sehr ähnlich, dann können $\mathcal{C}_1$ und $\mathcal{C}_3$ nicht komplett unähnlich sein. Des weiteren ist der Wert $Q_{VI}$ alleine von der Größenverteilung der Cluster abhängig, nicht aber von der Anzahl $n$ der Objekte. Daher lässt sich für verschieden große Datensätze die Qualität des heuristischen SAHN-Clusterverfahrens miteinander vergleichen.

Für $Q_{VI}$ gilt wie für die anderen vorgestellten Qualitätsmaße, dass der Erwartungswert nicht unabhängig von der Clusteranzahl $k$ ist. Daher wird im Folgenden das normalisierte Distanzmaß $Q_{NVI}$ (vgl. [VEB10]) verwendet. Es ist ebenfalls metrisch und unabhängig von der Objektanzahl $n$. Die Werte verschiedener Level (unterschiedliche Clusteranzahl) sind für das normalisierte Maß ebenfalls miteinander vergleichbar.

**6.1.7 Definition (Normalisiertes VI ($Q_{NVI}$)).**

$$Q_{NVI}(\mathcal{C}_1, \mathcal{C}_2) = 1 - \frac{I(\mathcal{C}_1, \mathcal{C}_2)}{H(\mathcal{C}_1, \mathcal{C}_2)} \tag{6.19}$$

$$H(\mathcal{C}_1, \mathcal{C}_2) = H(\mathcal{C}_1) + H(\mathcal{C}_2) - I(\mathcal{C}_1, \mathcal{C}_2) \tag{6.20}$$

## 6.2. Implementierung eines Evaluationsframeworks

Um die einzelnen Testfälle flexibel zusammenstellen zu können, wurde ein Evaluationsframework in der Software Scaffold Hunter implementiert. Es unterstützt die einfache Modellierung von Testserien und verfügt über einen modularen Aufbau, der die unkomplizierte Implementierung neuer Evaluationsmethoden, Qualitätsmaße und Testabläufe ermöglicht.

Die Erstellung der Testdatensätze erfolgt direkt in der Software Scaffold Hunter, denn diese unterstützt bereits das Importieren von Datensätzen aus unterschiedlichen Datenquellen. Es wäre unnötig, diese Funktionalität erneut zu implementieren. Daher kann für das Evaluationsframework ein eigenes Datenbankschema angelegt werden, in das die Testdatensätze über den Scaffold Hunter importiert, weiteren Eigenschaften (z. B. Fingerprints) berechnet und weitere Attribute aus synthetischen Quellen (z. B. RapidMiner[1]) hinzugefügt werden können. Die Sitzungsverwaltung des Scaffold Hunter kann anschließend dazu verwendet werden, Teilmengen der Datensätze zu definieren, die getestet werden sollen. Dazu wird eine neue Sitzung im Scaffold Hunter erstellt und der Datensatz anschließend mit Hilfe von Filtern und dem Teilmengenkonzept (vgl. Abschnitt 2.2) unterteilt. Die Tests werden schließlich auf allen definierten Teilmengen ausgeführt.

Um das Verständnis dieses Vorgehens zu verbessern, wird im Folgenden ein Beispiel beschrieben. Nehmen wir den hypothetischen Fall an, dass das Laufzeitverhalten eines Clusterverfahrens auf einem Datensatz aus der PubChem-Datenbank mit dem Tanimoto-Distanzmaß und dem DayLight-Fingerprint untersucht werden soll. Um dies zu erreichen, muss nun folgendermaßen vorgegangen werden: Zuerst wird der Datensatz aus PubChem im Scaffold Hunter importiert. Anschließend wird das DayLight-Berechnungsplugin verwendet, um aus den Molekülstrukturen Fingerprints zu berechnen. Nach der Berechnung der Fingerprints ist der Datensatz vollständig vorbereitet, so dass nun eine neue Sitzung angelegt und ausgeführt werden kann. Innerhalb der Sitzung werden jetzt verschiedene Teilmengen des Datensatzes mit unterschiedlichen Größen definiert. Nun kann der Scaffold Hunter geschlossen und das Evaluationsframework verwendet werden, um eine Performanztestserie zu starten. Das Clustering wird auf jeder Teilmenge einmal berechnet und die Performanzdaten anschließend in einer CVS-Datei gespeichert. Aus diesen Daten kann schließlich mit Hilfe eines externes Programms (z. B. GnuPlot) ein Diagramm für des Laufzeitverhalten erstellt werden.

Herzstück des Evaluationsframeworks sind die Implementierungen der abstrakten Klasse *EvaluationModule* (vgl. Abbildung 6.3). Sie stellen einzelne Testfälle dar. Implementiert wurde ein Performanz-Testfall (*PerformanceModule*) und ein Modul für den Vergleich zweier Clusterings (*ComparisonModule*). Beide Module lassen sich über Enums parametrisieren. Es kann das verwendete Clusterverfahren, die Nächste-Nachbar-Suchstrategie inkl. deren Parametrisierung, das Linkage-Verfahren und das Distanzmaß ausgewählt werden.

[1] http://rapid-i.com/

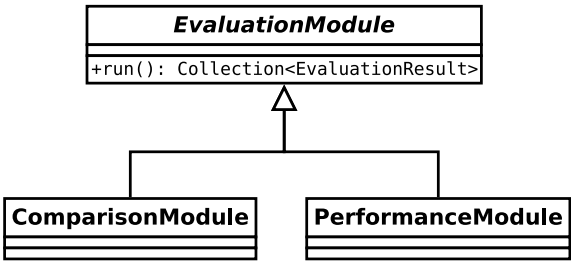



**Abbildung 6.3.:** Evaluationsmodule

Die verwendeten Eigenschaften der Moleküle werden automatisch passend zum Distanzmaß ermittelt.

Für das *ComparisonModule* können alle Parameter selbstverständlich für beide Seiten des Vergleichs festgelegt werden. Jede beliebige Kombination von Parametern ist daher miteinander vergleichbar. Ein weiterer Parameter, welcher ausschließlich für das *ComparisonModule* existiert, ist die Wahl eines Qualitätsmaßes (vgl. Abschnitt 6.1). Die Qualitätsmaße werden als Implementierung der abstrakten Klasse *HierarchicalComparison* realisiert (vgl. Abbildung 6.4). Die Implementierung *JaccardComparison* wurde in dieser Arbeit nicht verwendet, ist aber trotzdem in Abbildung 6.4 aufgeführt, da sie Teil der Implementierung ist.

Da alle implementierten Qualitätsmaße eine kubische Laufzeit besitzen, falls sie für jedes der $n$ Level berechnet werden, kann über einen weiteren Parameter die Schrittweite der Messungen eingestellt werden (z. B. jedes 50. Level). Des Weiteren wird die Berechnung der Qualitätsmaße für die unterschiedlichen Level parallel ausgeführt. Das Gleiche gilt für die Berechnung der Clusterings mit unterschiedlichen Parametern und Teilmengengrößen. Im letzten Fall sind der Parallelisierung jedoch Grenzen bzgl. des Speicherbedarfs gesetzt, da immer mit einem exakten Clusterverfahren verglichen wird, welches eine Distanzmatrix benötigt. Deshalb werden für die Evaluation maximal zwei Clusterverfahren parallel ausgeführt.

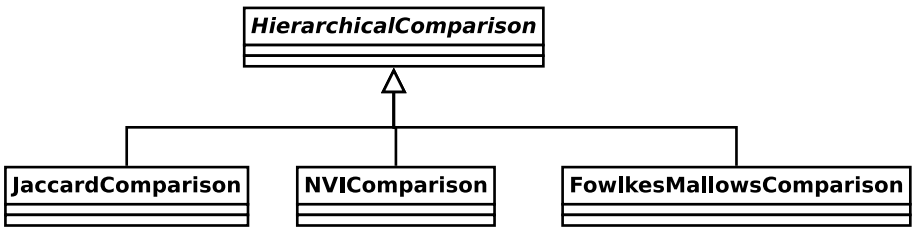



**Abbildung 6.4.:** Qualitätsmaße

Ist ein *EvaluationModule* initialisiert worden, kann es über die Run-Methode gestartet werden. Die Ergebnisse werden in Objekten des Typs *EvaluationResult* gespeichert. Für jeden Testlauf (z. B. für jede Teilmenge) wird ein eigenes Ergebnis angelegt. In jedem Ergebnis sind Metainformationen, wie z. B. die Parametereinstellungen, die verwen-

deten Eigenschaften der Moleküle, die Bezeichnung der Teilmenge, die Teilmengengröße und das verwendete Qualitätsmaß (z. B. *NVI Comparison*) festgehalten. Zudem werden die Ergebnisse der Testläufe als Schlüssel-Wert-Paare gespeichert. Listing 6.1 zeigt eine Beispielausgabe mittels *toString()*-Methode der *EvaluationResult*-Klasse.

```
Session: different_subset_sizes_estate
Dataset: pubchem 13500 - EState
Subset: Root
Subset size: 11604
Measurement: time performance-average of 3 values
Clustering 0: Generic
NN search strategy 0: Vorwärts NN search
NN search parameters 0: no parameter
Linkage 0: Centroid Linkage
Distance 0: Euclide
Used Properties 0: random-35,random-34,random-37,random-36,random-39

key,value
time to initialise,158414.66666666666
time to run,60102.0
overall time,218516.66666666663
```

**Listing 6.1:** Beispiel für eine Ausgabe der Klasse *EvaluationResult*

Mit Hilfe der Evaluationsmodule können somit leicht eine Vielzahl verschiedener Tests modelliert werden. Häufig existieren jedoch Anwendungsfälle, in denen das Ergebnis weiterverarbeitet werden soll oder in denen Tests mehrfach ausgeführt werden müssen. Für diesen Zweck wurde das Interface *EvaluationMetaModule* modelliert (vgl. Abbildung 6.5).

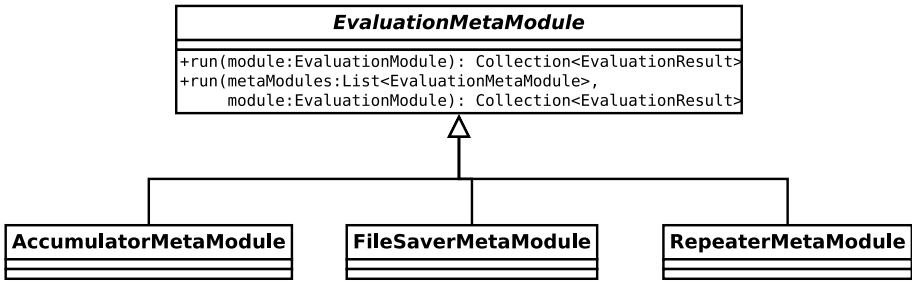



**Abbildung 6.5.:** EvaluationMetaModule

Metamodule können seriell hintereinander geschaltet werden. Dadurch lassen sich verschiedene Metamodule miteinander kombinieren. Implementiert wurden drei Metamodule: Das *RepeaterMetaModule* wiederholt einen Test mehrfach. In Kombination mit dem *AccumulatorMetaModule* können so z. B. durchschnittliche Laufzeiten für wiederholte Testläufe berechnet werden. Das *AccumulatorMetaModule* verschmilzt alle Tests bei denen die Metainformationen des *EvaluationResult* übereinstimmen. Für gleichwertige Metainformationen

wird dann eine Akkumulationsfunktion (z. B. Durchschnitt, Maximum, etc.) auf den Ergebnissen angewendet.

Ein weiteres Metamodul ist das *FileSaverMetaModule*. Es erlaubt das Abspeichern der Testergebnisse in einer oder mehreren Textdateien. Zusätzlich kann eine kommaseparierte CSV-Datei ohne Metainformationen gespeichert werden, um die Daten leicht in externen Programmen verarbeiten zu können. Häufig entstand in diesem Zusammenhang das Problem, dass die Ergebnisse aus mehreren Test in Bezug auf die Teilmengengröße verglichen werden sollten. Für eine Laufzeitanalyse ist es z. B. interessant, die Laufzeit im Vergleich zur Eingabegröße bzw. Teilmengengröße zu plotten. Deshalb unterstützt das *FileSaverMetaModule* die Reorganisation mehrerer Ergebnisse in Bezug auf die Teilmengengröße. Die erste Spalte der ausgegebenen CSV-Datei stellt dann die Teilmengengrößen dar und für jeden Schlüsselwert wird eine eigene Spalte generiert. Für einen Laufzeittest würde eine Zeile der CSV-Datei dann die Subsetgröße, die Laufzeit zur Initialisierung der *NNSearch*-Datenstruktur, die Laufzeit des Algorithmus und die Gesamtlaufzeit enthalten.

## 6.3. Testumgebung

Alle Tests wurden auf einem Linux-System mit folgender Hardware durchgeführt:

**CPU:** Intel Core2 Quad CPU Q9450 @ 2.66GHz (4 Kerne)

**Speicher:** 8 GiB, PC 6400, CL5, Dual Channel

**Chipsatz:** Intel X48, ICH9

Die Software des Systems wurde folgendermaßen gewählt:

**Betriebsystem:** Gentoo Linux 64 Bit (Stable Release: 04.08.2012)

**Kernel:** 3.3.8 (Gentoo Patchset), Preemptive CFQ Scheduler

**Java:** IcedTea 6.1.11.3

Der Heap Space der Java Virtual Machine wurde auf 5 GiB Speicher beschränkt (Kommandozeilenargument: `-Xmx5G`)) um jegliches Auslagern des Hauptspeichers auszuschließen. Alle Clustering-Algorithmen verwenden nur einen Thread und sind nicht parallelisiert. Es kann jedoch nicht ausgeschlossen werden, dass einzelne Bestandteile der Java-VM (z. B. die Garbage Collection G1) mehr als einen Kern verwenden.

## 6.4. Testdesign

Das Testdesign umfasst die Auswahl von Testdatensätzen, Algorithmen und der Parametrisierungen der Algorithmen, sowie die Modellierung von Testabläufen. Das Testdesign

wurde so gewählt, dass es die Bewertung von Qualität und Laufzeit des heuristischen SAHN-Clusterverfahrens in der Praxis erlaubt. Zusätzlich sollen passende Parametereinstellungen gefunden werden, mit denen eine möglichst gute Qualität bei geringer Laufzeit erzielt werden kann. Da die Parameterwahl vom verwendeten Datensatz abhängt, soll soweit möglich ein Konzept entwickelt werden, um die Parameter automatisch zu wählen oder die Komplexität der Parameterwahl für den Benutzer des Scaffold Hunter möglichst einfach zu gestalten. Zudem soll für die Beschränkung der maximalen Suchtiefe der Best-Frontier-Suche festgestellt werden, ob ein logarithmischer Schwellenwert bei gleichbleibender Qualität festgelegt werden kann. Dies hat Auswirkungen auf die asymptotische Laufzeit des heuristischen SAHN-Clustering-Algorithmus (vgl. Abschnitt 5.3.3.1)

### 6.4.1. Datensätze und Merkmalsselektion

Als Testdatensätze wurden sowohl synthetische, als auch reale Datensätze verwendet. Da die Repräsentation der Daten abhängig vom verwendeten Distanzmaß ist, werden die Datensätze hier nach dem Distanzmaß sortiert vorgestellt.

#### 6.4.1.1. Euklidisches Distanzmaß

Für das euklidische Distanzmaß wurden zwei Klassen von synthetischen Datensätzen zusammengestellt.

Die erste Klasse besteht aus Datensätzen mit gleichverteilten, zufälligen Werten. Die Datensätze wurden mit unterschiedlicher Dimensionalität $(2, 5, 10, 20)$ generiert[2]. Mit diesen Datensätzen soll der Einfluss der Dimensionalität auf den Pivot-Ansatz untersucht werden, da der Erwartungswert für die intrinsische Dimensionalität mit der euklidischen Dimensionalität identisch ist.

Der zweiten Klasse gehört nur ein einziger Datensatz mit der Dimensionalität 5 und 32 synthetischen, gaußverteilten Clustern an. Die einzelnen Cluster sind gut separiert; d. h. die Intra-Cluster-Distanzen des 0.9-Quantils der Gaußverteilung betragen maximal $\frac{1}{5}$ der Inter-Cluster-Distanzen. Mit Hilfe dieser Klasse wird der Einfluss der Clusterseparation auf die Clusteringqualität untersucht werden.

#### 6.4.1.2. Tanimoto-Distanzmaß

Für das Tanimoto-Distanzmaß wurde der Pyruvate-Kinase-Datensatz ausgewählt. Er wurde bereits in [WKR+09] verwendet, um die Scaffoldbaumansicht im Scaffold Hunter zu evaluieren. Für diesen Datensatz wurden der EState- sowie der DayLight-Fingerprint berechnet. Die Fingerprints verfügen über 79 bzw. 1024 Bits und werden in der Medikamenteforschung verwendet. Der Pyruvate-Kinase-Datensatz ist ein nicht-synthetischer

[2] Als Begründung/Rechtfertigung, warum keine Daten mit höherer Dimensionalität getestet wurden, sei auf die Ergebnisse in Abschnitt Abschnitt 6.5 verwiesen.

Datensatz und damit sehr relevant, um die Praxistauglichkeit des heuristischen SAHN-Clusterverfahrens zu untersuchen.

### 6.4.2. Algorithmen und Parametrisierung

Im Rahmen der aktuellen Implementierung stehen sich der NNChain-Algorithmus sowie der Generic-Clustering-Algorithmus zum Vergleich gegenüber. Zusätzlich kann noch eine Nächste-Nachbar-Suchstrategie gewählt werden.

Für den NNChain-Algorithmus können die exakten Nächste-Nachbar-Suchstrategien (matrix- und repräsentantenbasierte Nächste-Nachbar-Suche) verwendet werden. Da bereits in früheren Messungen die Laufzeit der matrixbasierten Nächste-Nachbar-Suche der repräsentantenbasierten Nächste-Nachbar-Suche überlegen war und hierfür auch eine plausible Erklärung existiert (vgl. Kapitel 3), wird lediglich die matrixbasierte Nächste-Nachbar-Suche verwendet. Ein weiterer Grund, die repräsentantenbasierte Nächste-Nachbar-Suche nicht zu verwenden, ist die dadurch resultierende Beschränkung auf spezielle Kombination von Linkage-Verfahren und Distanzmaß.

Der modifizierte Generic-Clustering-Algorithmus funktioniert mit allen Nächste-Nachbar-Suchstrategien. Die matrixbasierte Nächste-Nachbar-Suche kann durch die Vorwärts-Version ersetzt werden. Dies bietet einen Laufzeitvorteil, ohne den Speicherbedarf zu erhöhen oder heuristische Ergebnisse zu produzieren. Die repräsentantenbasierte Nächste-Nachbar-Suche wird aus den gleichen, bereits im letzten Absatz genannten, Gründen verworfen. Schließlich kann noch die heuristische Nächste-Nachbar-Suche mittels Best-Frontier-Suche verwendet werden.

Zusätzlich zu den Algorithmen und den Nächste-Nachbar-Suchstrategien müssen die Parameter gewählt werden, welche getestet werden sollen. Für die matrixbasierten Nächste-Nachbar-Suchen existieren keine Parameter. Die Best-Frontier-Suche in Kombination mit dem Pivot-Baum kann über drei Werte parametrisiert werden: Der Pivot-Baum besitzt einen Parameter $f$ für die Anzahl der Pivots pro Knoten und einen Parameter $l$ für die maximale Blattanzahl und die Best-Frontier-Suche kann über die Beschränkung der maximalen Suchtiefe $s$ parametrisiert werden.

Folgende Parameterkombinationen werden für den Pivot-Baum verwendet, wobei die Tupel dem Schema $(f, l)$ folgen: $(20, 1)$, $(50, 1)$, $(5, 10)$, $(5, 20)$ $(5, 50)$, $(10, 10)$.
Die beiden ersten Parametereinstellungen entsprechen der Verwendung der Best-Frontier-Suche ohne Pivot-Baum. Durch Vergleich der Ergebnisse der ersten beiden mit den restlichen Parametereinstellungen, kann daher die Effektivität des Pivot-Baumes ermittelt werden.

Für jede dieser Parametereinstellungen des Pivot-Baumes wird der Parameter $s$ in den folgenden Ausprägungen untersucht: 500, 1000, 2000, 4000, 8000, 16000, *unbegrenzt*.

### 6.4.3. Testszenarien

Bei Vergleichen zwischen dem NNChain-Algorithmus und dem heuristischen SAHN-Clusterverfahren ergibt sich ein spezielles Problem: Die beiden Verfahren unterstützen nicht ein einziges gemeinsames Linkage-Verfahren.

Daher wird die Performanz zwischen dem exakten NNChain-Clustering (matrixbasierte Nächste-Nachbar-Suche) und dem exakten Generic-Clustering-Algorithmus nur für einen Testdatensatz und ein gemeinsames Linkage-Verfahren verglichen. Die Laufzeit dieses Testlaufs für den Generic-Clustering-Algorithmus wird im Folgenden als Referenzlaufzeit bezeichnet. Anschließend wird ausschließlich die exakte Nächste-Nachbar-Suche mit der heuristischen Nächste-Nachbar-Suche in Kombination mit dem Generic-Clustering-Algorithmus verglichen. Dieses Vorgehen lässt sich folgendermaßen begründen: Der NNChain-Algorithmus ist in der Laufzeit vom verwendetem Linkage-Verfahren und der Struktur des Datensatzes unabhängig, da in jedem Fall exakt gleich viele Nächste-Nachbar-Suchen und Clusterverschmelzungen durchgeführt werden müssen. In Bezug auf den Generic-Clustering-Algorithmus ist diese Aussage nicht gültig. Die Gründe dafür sind in Abschnitt 4.1.3.3 bereits ausführlich behandelt worden. Die Laufzeitunterschiede des Generic-Clustering-Algorithmus können jedoch zwischen den einzelnen Testläufen direkt miteinander verglichen und immer auf die Referenzlaufzeit bezogen werden. Daher können die Laufzeiten von dem NNChain-Algorithmus und dem heuristische SAHN-Clusterverfahren dennoch miteinander verglichen werden.

Für die Qualitätsvergleiche stellt die oben aufgeführte Einschränkung bzgl. der Linkage-Verfahren kein Problem dar: Wenn beide Clusterverfahren korrekt implementiert sind, produzieren sie auch die selben Ergebnisse. Ausnahmefälle entstehen wenn aufgrund identischer Distanzen der nächste Nachbar nicht eindeutig ermittelt werden kann. In diesen Fällen ist jedoch auch das Ergebnis eines speziellen Verfahrens einem gewissen Nichtdeterminismus unterworfen. Daher kann der Generic-Clustering-Algorithmus immer anstatt des NNChain-Algorithmus gewählt werden, um die Qualität des heuristischen Verfahrens zu ermitteln. Um die Korrektheit der Implementierungen zu bestätigen, wurden darüber hinaus Qualitätsvergleiche der exakten Clusterverfahren für gemeinsame Linkage-Verfahren vorgenommen.

Das Vorgehen lässt sich daher zusammenfassend wie folgt beschreiben: Für jeden Testdatensatz (in Kombination mit dem Distanzmaß) und jede Parametereinstellung wird eine Performanzmessung und ein Qualitätsvergleich zwischen exakten und heuristischen Verfahren durchgeführt. Außer für die oben beschriebenen Spezialfälle, wird dafür ausschließlich der Generic-Clustering-Algorithmus verwendet. Im exakten Fall wird dieser mit der Vorwärts-Nächste-Nachbar-Suche und im heuristischen Fall mit der Best-Frontier-Suche kombiniert.

Es werden unterschiedliche Testabläufe für die Performanzmessungen und die Qualitätsmessungen verwendet. Diese sind in den folgenden Abschnitten festgehalten.

#### 6.4.3.1. Performanzmessungen

Die Performanzmessungen werden viermal wiederholt und anschließend wird der Durchschnitt der letzten drei Messungen berechnet. Die erste Messung geht daher nicht in das Ergebnis ein. Die Begründung dafür ist, dass sich für den ersten Testlauf häufig Ausreißer in der Laufzeit ergeben haben. Diese sind wahrscheinlich durch Caching-Effekte hervorgerufen worden. Die Unterschiede waren dabei stets deutlich kleiner als 10%. Trotzdem würde der Laufzeitplot ansonsten leicht einen falschen Eindruck vermitteln: Die größte Teilmenge eines Datensatzes wird aus Gründen der zugrundeliegenden Datenbankstruktur immer zuerst getestet. Daher würde ein Knick in der Laufzeit am rechten Rand des Plots schnell als zu schlechte oder zu gute asymptotische Laufzeit interpretiert.

Die Laufzeiten setzen sich bei jedem Test aus den Laufzeiten für die Initialisierung der Datenstruktur und dem Clustering selbst zusammen. Für die exakten Nächste-Nachbar-Suchen beinhaltet die Initialisierung daher die Zeit für die Berechnung der Distanzmatrix. Bei der heuristischen Nächste-Nachbar-Suche wird die Initialisierung des Pivot-Baumes und der Distanzlisten gemessen.

#### 6.4.3.2. Qualitätsmessungen

Die Qualitätsmessungen werden aus Zeitgründen nur einfach ausgeführt, da die Performanz der Qualitätsmaße nicht ausreichte, um mehrere Messungen innerhalb eines praktikablen Zeitintervalls durchzuführen. Für den Vergleich der Clusteringqualität werden immer der Fowlkes-Mallows-Index und das NVI Qualitätsmaß verwendet.

### 6.4.4. Abschließende Bemerkungen zum Testdesign

Die Performanz der Qualitätsmessungen hat sich in Kombination mit der großen Parameteranzahl als limitierender Faktor der Evaluation herausgestellt. Daher wurden die Datensätze in ihrer Größe reduziert. Auf dem in Abschnitt 6.3 beschriebenen Testsystem benötigt die Ausführung aller Tests für einen einzelnen Datensatz mit 12000 Objekten in etwa eine Woche. Bei sieben Datensätzen ergibt sich daher eine Laufzeit von ein bis zwei Monaten! Um detaillierte Erkenntnisse aus den Ergebnissen extrahieren zu können, wäre es wünschenswert weitere Parametereinstellungen oder Datensätze in den Tests zu berücksichtigen. Aus den soeben genannten Gründen ist dies jedoch nicht möglich gewesen.

## 6.5. Ergebnisse

Da es nicht möglich ist, alle Forschungsfragen auf einmal zu untersuchen, wird dieser Abschnitt Schritt für Schritt einzelne Aspekte der Testergebnisse aufgreifen und am Ende zu einer Gesamtaussage zusammenfassen.

Wenn im Folgenden von hohen oder niedrigen Leveln gesprochen wird, dann bezeichnet diese Höhenangabe die Höhe des horizontalen Schnittes im Dendrogramm. Der Schnitt des Levels 4 liegt beispielsweise über dem des Levels 6. Auch wenn die Level in den Plots von links nach rechts aufsteigend sortiert sind, sollte stets beachtet werden, dass die Verschmelzungsreihenfolge genau umgekehrt stattfindet. Ein Folgefehler einer Fehlentscheidung auf Level $x$ kann daher nur auf einem Level $y < x$ vorkommen.

### 6.5.1. NNChain vs. Generic Clustering

Wie in Abschnitt 6.4.3 erläutert, wird in Abbildung 6.6 eine Referenzlaufzeit des modifizierten Generic-Clustering-Algorithmus in Bezug auf den NNChain-Algorithmus dargestellt. Alle Messungen wurden mit Average-Linkage auf dem EState-Datensatz durchgeführt. Da die Laufzeit der Matrixinitialisierung bei beiden Algorithmen identisch ist, wurde ausschließlich die Laufzeit des Clusterverfahrens gemessen, welches die fertig berechnete Distanzmatrix als Eingabe erhält. Die Komplexität des Distanzmaßes geht damit an keiner Stelle in die Messungen mit ein.

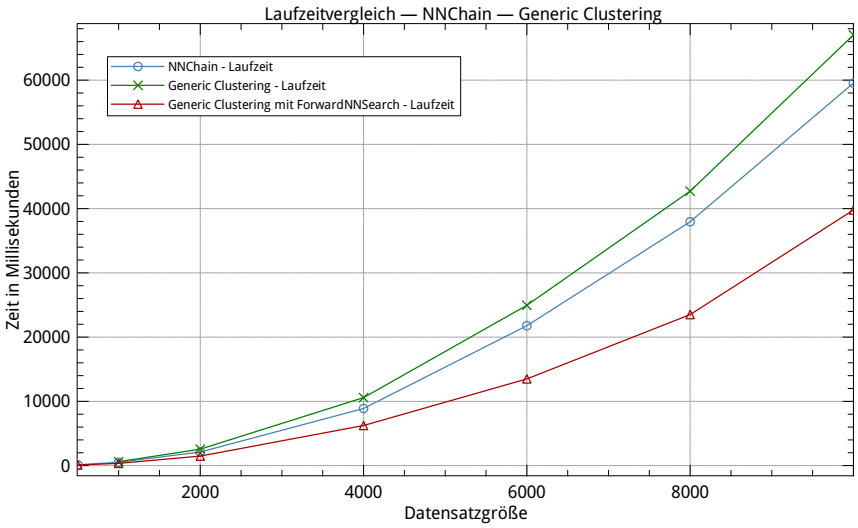



**Abbildung 6.6.:** Laufzeitvergleich: NNChain-Algorithmus, Generic-Clustering-Algorithmus

Für den Generic-Clustering-Algorithmus wurde sowohl die Laufzeit in Kombination mit der Vorwärts-Nächste-Nachbar-Suchstrategie, als auch die Laufzeit in Kombination mit der klassischen Matrix-Nächste-Nachbar-Suchstrategie gemessen. Es sei an dieser Stelle noch

einmal darauf hingewiesen, dass die Vorwärts-Nächste-Nachbar-Suchstrategie prinzipiell der Matrix-Nächste-Nachbar-Suchstrategie vorzuziehen ist. Die Matrix-Nächste-Nachbar-Suchstrategie wurde daher nur gemessen, um die Laufzeitverbesserung der Vorwärts-Nächste-Nachbar-Suchstrategie beurteilen zu können.

Bei der Analyse von Abbildung 6.6 zeigt sich, dass alle gemessenen Laufzeiten dem Verlauf einer quadratischen Funktion folgen. Der modifizierte Generic-Clustering-Algorithmus in Kombination mit der matrixbasierten Nächste-Nachbar-Suche ist ungefähr um den Faktor 1,2 langsamer als der NNChain-Algorithmus, während die Generic-Clustering-Variante in Kombination mit der Vorwärts-Nächste-Nachbar-Suchstrategie nur etwa zwei drittel der Laufzeit des NNChain-Algorithmus benötigt. Damit ist der modifizierte Generic-Clustering-Algorithmus dem NNChain-Algorithmus bzgl. der Laufzeit überlegen. Dieses Ergebnis widerspricht den Messungen aus [Mü11], in denen der Generic-Clustering-Algorithmus etwas langsamer als der NNChain-Algorithmus ist. In diesen Messungen wurde jedoch der unmodifizierte Algorithmus verwendet und die Verbesserung der Laufzeit kann daher auf die Modifikationen oder auf Implementierungsdetails zurückzuführen sein.

### 6.5.2. Qualität und Performanz in der Praxis

Im Folgenden wird die Qualität des Clusterings bzgl. verschiedener Parametereinstellungen für den Pyruvate-Kinase-Datensatz in Kombination mit dem EState- und DayLight-Fingerprint betrachtet und anschließend mit dem Laufzeitverhalten in Verbindung gebracht. Anschließend werden Parametereinstellungen extrahiert, für welche Laufzeit und Qualität in einem guten Verhältnis stehen.

#### 6.5.2.1. Einfluss der Pivot-Anzahl und der Pivot-Baum-Tiefe

In diesem Abschnitt wird auf die Beschränkung der maximalen Suchtiefe verzichtet und ausschließlich die Pivot-Anzahl und die maximale Blattanzahl variiert.

Abbildung 6.7 zeigt die mit dem Fowlkes-Mallows-Index gemessene Clusteringqualität für den EState-Datensatz. Die Clusterings aller Parametereinstellungen zeigen einen Qualitätseinbruch in den untersten Leveln und eine gleichmäßig gute Qualität um das Level 1750. Oberhalb dieses Levels gibt es ebenfalls einen Qualitätseinbruch für alle Einstellungen.

Gut erkennbar ist der Zusammenhang zwischen der Qualität und der Pivot-Anzahl im Bereich der Level 2000 bis 10000. Clusterings, welche eine hohe Pivot-Anzahl verwenden, schneiden im Allgemeinen besser ab. Einen Ausreißer stellt die Qualitätsmessung für die Parametereinstellung mit einem Verzweigungsgrad von 5 und einer maximalen Blattanzahl von 10 dar. Es ist jedoch nicht verwunderlich, dass die einzelnen Testläufe einer gewissen Zufälligkeit unterliegen, da die Pivots mittels einer zufälligen Stichprobe gezogen werden.

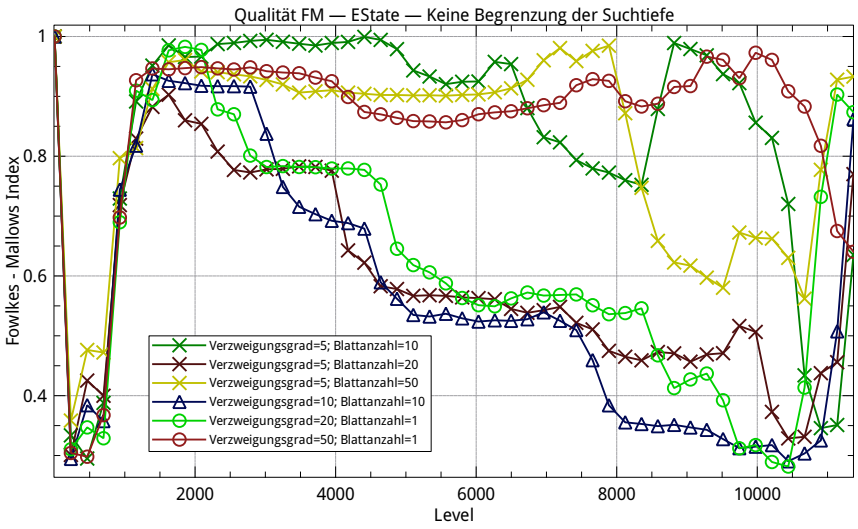



**Abbildung 6.7.:** Qualität FM; EState-Datensatz; keine Begrenzung der Suchtiefe

Auffallend ist, dass die Verschmelzungen der unteren Level (8000 bis 11000) eine Art kritischen Punkt für die Clusteringqualität der mittleren Level (2000 bis 8000) darstellen. Fallen in den unteren Leveln die richtigen Entscheidungen, dann liegt die Clusteringqualität für die mittleren Level relativ konstant zwischen 0,8 und 1,0. Werden hingegen die falschen Entscheidungen getroffen, dann steigt die Qualität nur allmählich von 0,4 auf 0,9 (bei Level 1750) an. Im Allgemeinen scheinen die Kurven für kleinere Abschnitte immer einem Trend zu folgen, d. h. es findet sich kein verrauschtes *Zick-Zack*-Muster, sondern gleichmäßige Verbesserungen oder Verschlechterungen der Qualität. Dies legt die Vermutung nahe, dass es einige kritische und andere nicht so kritische Entscheidungen im Agglomerationsprozess gibt und kritische Fehlentscheidungen eine Reihe von Folgefehlern auslösen. Eine mögliche Erklärung für die gute Qualität um Level 1750 wird in Abschnitt 6.5.3.2 gegeben.

Im Gegensatz zum Fowlkes-Mallows-Index zeigt das NVI-Qualitätsmaß (vgl. Abbildung 6.8) deutlich weniger starke Schwankungen zwischen den verschiedenen Clusterings und es ist kein Qualitätseinbruch bei den unteren Leveln zu beobachten. Die Qualität scheint von Level 11000 bis Level 6000 linear abzufallen und dann, wie beim Fowlkes-Mallows-Index, bei Level 1750 wieder eine relativ gute Qualität von etwa 0.8 zu erreichen. In den sehr hohen Leveln ist genau wie beim Fowlkes-Mallows-Index ein Qualitätseinbruch zu erkennen. Das im letzten Absatz beschriebene Trendverhalten ist ebenfalls zu beobachten.

Der Plot für den DayLight-Fingerprint in Kombination mit dem Fowlkes-Mallows-Index (Abbildung 6.9) zeigt einige Parallelen und Unterschiede zum Plot des EState-Fingerprints auf. Gemeinsamkeiten lassen sich im Verlauf der Kurve finden: Sie fällt in den unteren Leveln schnell ab und erreicht etwa bei Level 1750 erneut ihren Höhepunkt. Ein Quali-

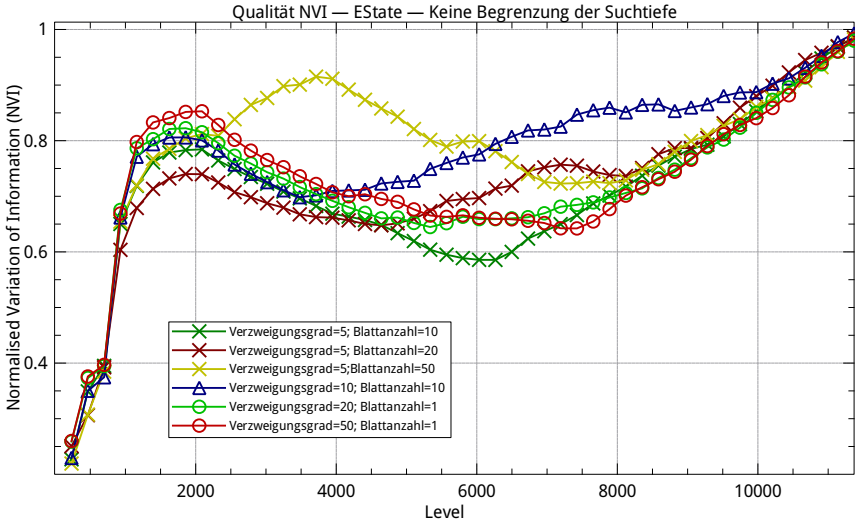



**Abbildung 6.8.:** Qualität NVI; EState-Datensatz; keine Begrenzung der Suchtiefe

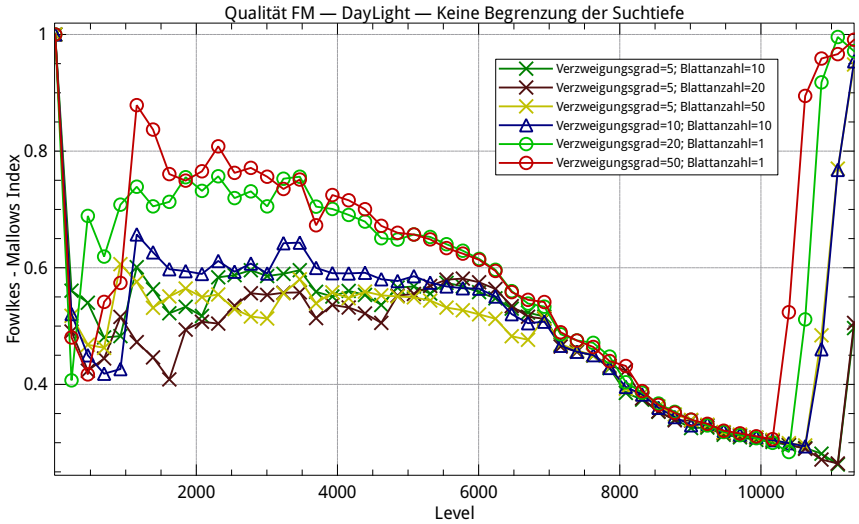



**Abbildung 6.9.:** Qualität FM; DayLight-Datensatz; keine Begrenzung der Suchtiefe

tätseinbruch für die hohen Level ist ebenfalls zu beobachten, fällt aber (relativ zum Level 1750) deutlich geringer aus. Absolut gesehen unterscheidet sich die Qualität im Vergleich zum EState-Datensatz in den hohen Leveln kaum. Es gibt keine Parametereinstellung, welche (wie beim EState-Fingerprint) direkt nach dem Qualitätsabfall in den unteren Leveln wieder über den Wert 0.8 steigt. Alle Einstellungen zeigen ein ähnliches Verhalten in den Leveln 6000 bis 10000 und differenzieren sich erst auf den höheren Leveln.

Wie zuvor erwähnt stellen die Parametereinstellungen mit $l = 1$ ($l$ entspricht der maximalen Blattanzahl) eine Best-Frontier-Suche ohne Pivot-Baum dar. Interessanterweise

schneiden diese Einstellungen besser als alle anderen ab, auch wenn die erwartete Anzahl von verwendeten Pivots im Pivot-Baum für die Einstellung $f = 10;\ l = 10$ der Pivot-Anzahl 20 entspricht. Es muss jedoch erwähnt werden, dass einige dieser Pivots nur für eine Teilmenge der Distanzabschätzungen verwendet werden und die reine Pivot-Anzahl nicht direkt miteinander vergleichbar ist. Andererseits könnte dieses Verhalten auch ein Hinweis darauf sein, dass die Verringerung der Baumtiefe durch die Einfügeoperationen (vgl. Abschnitt 5.3.2.1) des Pivot-Baumes schon auf relativ tiefen Leveln beginnt.

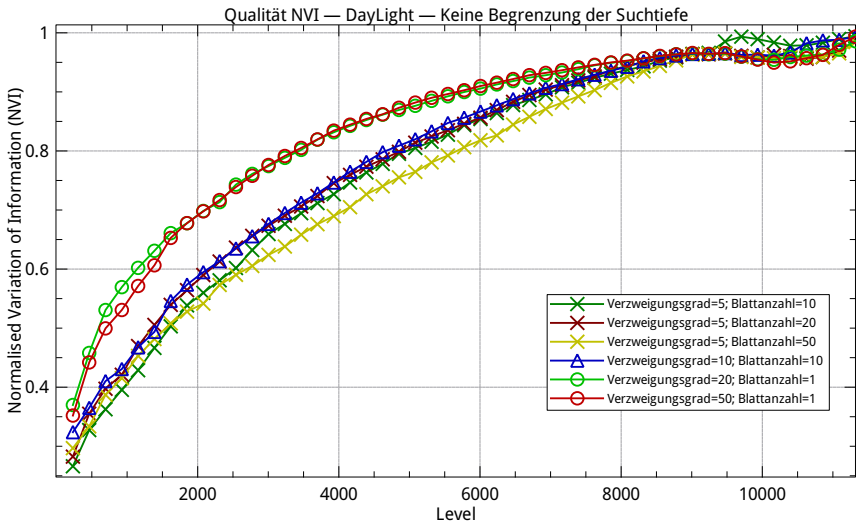



**Abbildung 6.10.:** Qualität NVI; DayLight-Datensatz; keine Begrenzung der Suchtiefe

Das NVI-Qualitätsmaß (Abbildung 6.10) zeigt erneut deutlich weniger Differenzen zwischen den Parametereinstellungen. Wie beim Fowlkes-Mallows-Index ist die Qualität der Clusterings mit $l = 1$ am besten. Unerwarteterweise schneidet die Parametereinstellung mit $f = 5;\ l = 50$, im Vergleich zu denen mit niedriger Blattanzahl, schlechter ab. Diese Abweichung ist jedoch sehr gering und kann somit zufällig zustande gekommen sein. In den sehr hohen Leveln liegt die Qualität dieser Parametereinstellung zudem wieder über den Einstellungen mit niedriger Blattanzahl.

Auffallend ist insbesondere der monotone Abfall der Qualität mit der Höhe der Level. Dies wirft die Frage auf, ob der zwischenzeitliche Anstieg der Qualität in Abbildung 6.9 allein durch die Eigenschaften des Qualitätsmaßes verursacht wurde (vgl. Abbildung 6.2). Der Qualitätsknick in den unteren Leveln kann ebenfalls durch diesen Einfluss hervorgerufen sein.

Des Weiteren stellt sich die Frage, ob sich der monotone Qualitätsabfall für größere Datensätze fortsetzt. Nach etwa 10000 Verschmelzungen könnten sich die Fehler soweit aufsummiert haben, dass die Qualität noch höherer Level unzureichend wird. In diesem Fall würde die Qualität des heuristischen SAHN-Clusterverfahren für große Datensätze

prinzipiell unzureichend sein. Aus den genannten Gründen (Performanz der Qualitätsmaße) kann dieser Zusammenhang für größere Datensätze hier nicht untersucht werden. Es ist jedoch möglich, einen Blick auf die Qualitätsmessungen für kleinere Datensätze zu werfen und interessanterweise zeigt sich für alle Datensatzgrößen dieselbe Kurve. Sie ist jedoch auf den entsprechenden Level-Bereich gestaucht. Zum Beispiel entspricht der Qualitätswert für das Level 1000 bei einer Datensatzgröße von 10000 in etwa dem Wert des Levels 100 bei einer Datensatzgröße von 1000. Daher ist anzunehmen, dass die Qualität des Clusterings auch für größere Datensätze vergleichbar mit den hier gezeigten Beispielen ist.

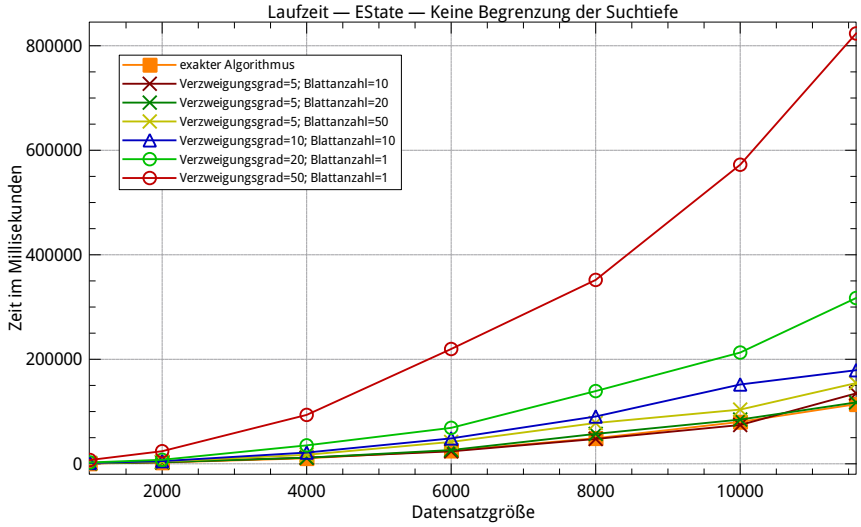



**Abbildung 6.11.:** Performanz; EState-Datensatz; keine Begrenzung der Suchtiefe

Nachdem zunächst die Qualitätsmessungen erläutert wurden, wird im Folgenden der Zusammenhang zwischen den Laufzeiten hergestellt. Unabhängig von der Wahl des Fingerprints besitzt das heuristische SAHN-Clusterverfahren in der Praxis eine quadratische Laufzeit (vgl. Abbildung 6.11 und Abbildung 6.12). Dies entspricht der theoretischen Analyse (vgl. Abschnitt 5.3.3.1), da die Suchtiefe der Best-Frontier-Suche nicht beschränkt wurde und damit $s = n$ gilt.

Das heuristische SAHN-Clusterverfahren zeigt im Vergleich mit dem exakten Algorithmus keine Verbesserungen der Laufzeit, falls der EState-Fingerprint verwendet wird. Bei Verwendung des DayLight-Fingerprints lässt sich hingegen eine Verbesserungen der Laufzeit um den Faktor zwei bis drei für einige Parametereinstellungen beobachten. Der DayLight-Fingerprint verwendet mehr als zehnmal so viele Bits wie der EState-Fingerprint und die Berechnung der Distanzen mit dem Tanimoto-Distanzmaß benötigt daher mehr Rechenzeit. Durch die Verwendung der Best-Frontier-Suche müssen jedoch nur noch linear viele Distanzberechnungen im Verhältnis zur Eingabegröße durchgeführt werden. Daher ist es nicht verwunderlich, dass der Laufzeitvergleich für das heuristische SAHN-Clusterverfahren

besser ausfällt, falls rechenintensive Distanzmaße verwendet werden. Die Gesamtlaufzeit des heuristischen SAHN-Clusterverfahrens ist somit weniger stark von der Rechenkomplexität des Distanzmaßes abhängig und das Laufzeitverhalten des Algorithmus entspricht in diesem Punkt der Zielvorgabe rechenintensive Distanzmaße besser zu unterstützen.

Bei beiden Fingerprints führt die Verwendung des Pivot-Baumes zu einer Verbesserungen der Laufzeit um einen konstanten Faktor. Interessanterweise unterscheidet sich die Laufzeit des heuristischen SAHN-Clustering-Algorithmus für unterschiedliche Wahl von $l$ (maximale Blattanzahl) kaum. Es erscheint daher auf den ersten Blick sinnvoll $l$ relativ groß zu wählen, um die Qualität zu steigern ohne die Laufzeit deutlich zu verschlechtern. Leider hat sich der Zusammenhang zwischen hoher Blattanzahl und hoher Qualität in den Qualitätsmessungen bisher nicht eindeutig belegen lassen und daher ist diese Aussage (zumindest für unbeschränktes $s$) nicht zu bestätigen.

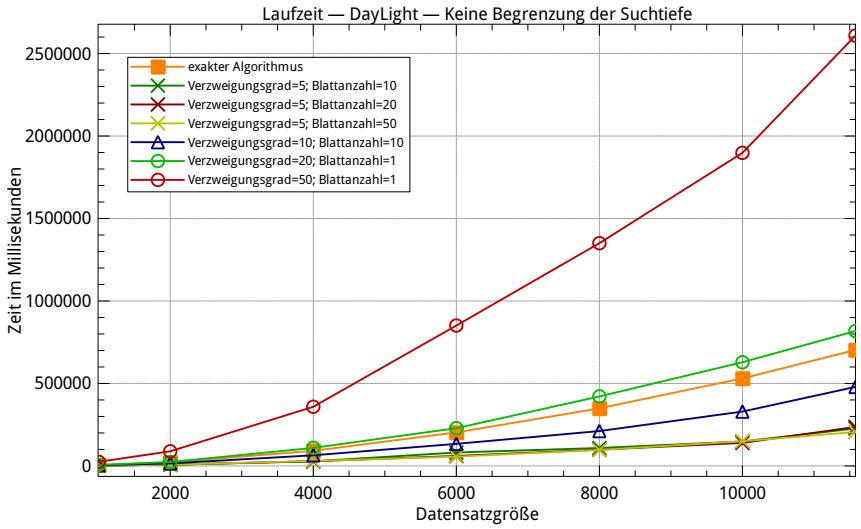



**Abbildung 6.12.:** Performanz; DayLight-Datensatz; keine Begrenzung der Suchtiefe

#### 6.5.2.2. Einfluss der Suchtiefe

Im Folgenden wird der Einfluss des Parameters $s$ nur für die Parametereinstellungen $f = 5;\ l = 10$ und $f = 5;\ l = 50$ und den DayLight-Datensatz genau untersucht, um den Umfang dieser Arbeit nicht zu überschreiten. Die Parametereinstellungen $f = 5;\ l = 10$ und $f = 5;\ l = 50$ wurden ausgewählt, da sie sich gut verwenden lassen, um den Einfluss von $l$ auf die mögliche Wahl des Parameters $s$ zu untersuchen. In den bisherigen Messungen zeigt der Fowlkes-Mallows-Index ein differenzierteres Bild als das NVI-Qualitätsmaß und das ist auch bei den hier vorgenommen Messungen der Fall. Da dieser Abschnitt nicht den Vergleich der Qualität unterschiedlicher Level zum Ziel hat, ist der Fowlkes-Mallows-Index am besten für die Analyse geeignet.

Auch wenn die Parametereinstellungen $f = 20;\ l = 1$ und $f = 50;\ l = 1$ an dieser Stelle nicht im Detail untersucht werden sollen, kann eine generelle Aussage in Bezug auf diese formuliert werden: Sie reagieren sehr sensibel auf eine Begrenzung der Suchtiefe und der Qualitätsverlust (bei der Wahl kleiner Werte für $s$) ist sehr hoch. Die Beobachtung lässt sich dadurch erklären, dass die Suchtiefe linear mit der Anzahl der Pivots steigen muss, um pro Distanzliste im Durchschnitt gleich tief zu suchen. Werden viele Pivots verwendet, muss daher auch die Suchtiefe erhöht werden.

Des Weiteren hat die Heuristik, welche im Falle des Erreichens der maximalen Suchtiefe angewandt wird (vgl. Abschnitt 4.2.3.2), einen Einfluss auf die Wahl des nächsten Nachbarn. Diese Heuristik wählt das Objekt als nächsten Nachbar, welches bisher am häufigsten gezählt wurde. Dadurch werden lokale Objekte als nächste Nachbarn bevorzugt, da für diese mehr Pivots verwendet werden und damit die Wahrscheinlichkeit steigt, häufiger gezählt zu werden. Lokale Objekte sind als nächste Nachbarn deutlich wahrscheinlicher als nicht-lokale-Objekte und daher hat dieser Effekt einen positiven Einfluss auf die Qualität.

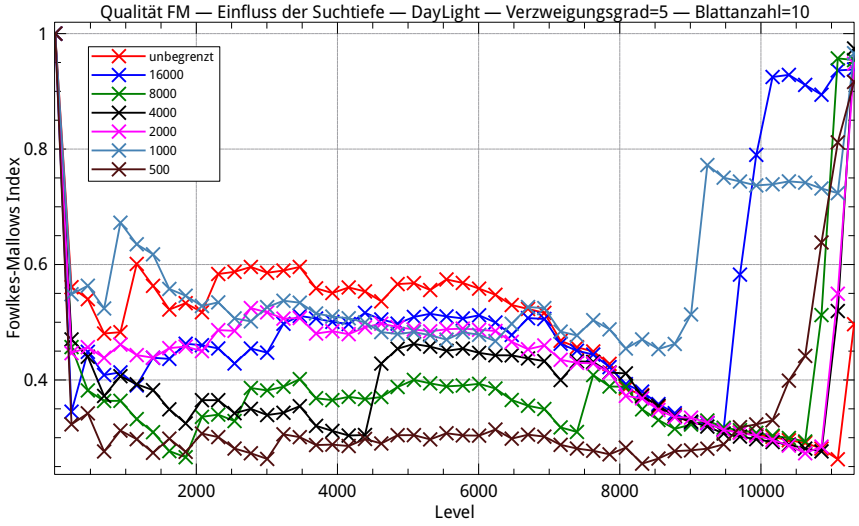



**Abbildung 6.13.:** Einfluss der Suchtiefe auf die Qualität; DayLight-Datensatz; f=5; l=10

Abbildung 6.13 und Abbildung 6.14 zeigen den Einfluss des Parameters $s$ auf die Qualität der Clusterings. Interessanterweise verschlechtert sich die Qualität bei einer großen Blattanzahl nicht für einen niedrigen Wert von $s$, obwohl die Suchtiefe mit der Anzahl der Pivots zunehmen müsste, um die Distanzlisten gleich tief zu durchsuchen (vgl. Argumentation des letzten Absatzes). Es tritt sogar der gegenteilige Effekt ein, dass eine Erhöhung von $l$ einen kleineren Wert von $s$ bei gleichbleibender Qualität ermöglicht. Eventuell hat der beschriebene Effekt der Heuristik, bei Erreichen der maximalen Suchtiefe, einen größeren Einfluss auf die Qualität der Nächste-Nachbar-Suche.

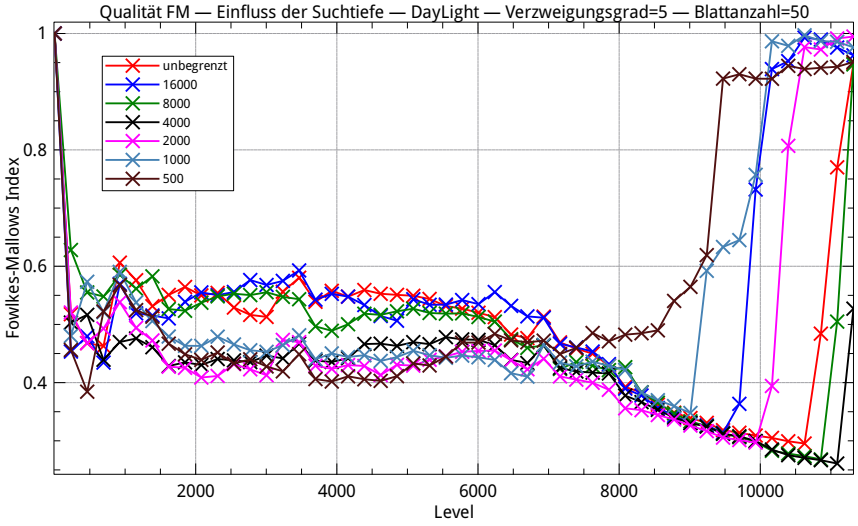



**Abbildung 6.14.:** Einfluss der Suchtiefe auf die Qualität; DayLight-Datensatz; f=5; l=50

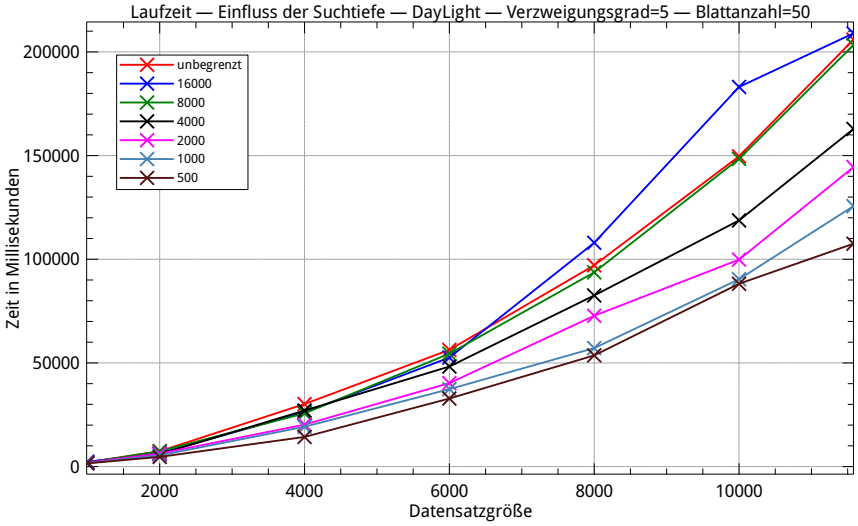



**Abbildung 6.15.:** Einfluss der Suchtiefe auf die Laufzeit; DayLight-Datensatz; f=5; l=50

Da die Laufzeiten beider Parametereinstellungen kaum Unterschiede aufzeigen, ist eine große Blattanzahl prinzipiell vorzuziehen. Die Abbildung 6.15 zeigt exemplarisch den Einfluss von $s$ auf die Laufzeit für die Parametereinstellung $f = 5$; $l = 50$. Für eine Datensatzgröße von 10000 und $s = 1000$ ist die Laufzeit in etwa siebenmal so schnell wie der exakte Algorithmus und die Qualität des Clusterings ist nur unwesentlich schlechter als bei einer unbegrenzten Suchtiefe. Für den DayLight-Fingerprint lässt sich daher zusammenfassen, dass eine hohe Blattanzahl mit der Wahl eines niedrigen $s$ zu guten Ergebnissen und geringer Rechenzeit führt.

Der EState-Fingerprint zeigt ein ähnliches Verhalten wie der DayLight-Fingerprint. Die Wahl eines niedrigeren Schwellenwertes für die Suchtiefe führt jedoch zu einer etwas stärkeren Reduzierung der Qualität und der stabilisierende Effekt der größeren Blattanzahl (auf die Qualität) ist nicht so deutlich ausgeprägt. Dies verwundert etwas, da für eine hohe Bit-Anzahl des Fingerprints auch eine höhere intrinsische Dimensionalität der Daten zu erwarten ist und eine hohe Dimensionalität eine Verminderung der Effektivität der Pivots bedeutet (vgl. Abschnitt 4.2.3).

Aus Abbildung 6.15 ist leider nicht abzulesen, ob die praktische asymptotische Laufzeit quadratisch oder subquadratisch ist. Deshalb wurde eine Messung der Laufzeit für die Parametereinstellung $s = 4000;\ f = 5;\ l = 50$, den Pyruvate-Kinase-Datensatz und den DayLight-Fingerprint bei einer Datensatzgröße von 50000 Objekten vorgenommen (Abbildung 6.16). Um eine Einschätzung der Performanz vornehmen zu können, wurde der Datensatz ebenfalls mit dem exakten Verfahren geclustert. Aus genannten Gründen kann die Qualität dieser Einstellung nicht geprüft werden. Die Wahl des Parameters $s$, welche an dieser Stelle recht willkürlich erscheint, wird in Abschnitt 6.5.4 legitimiert.

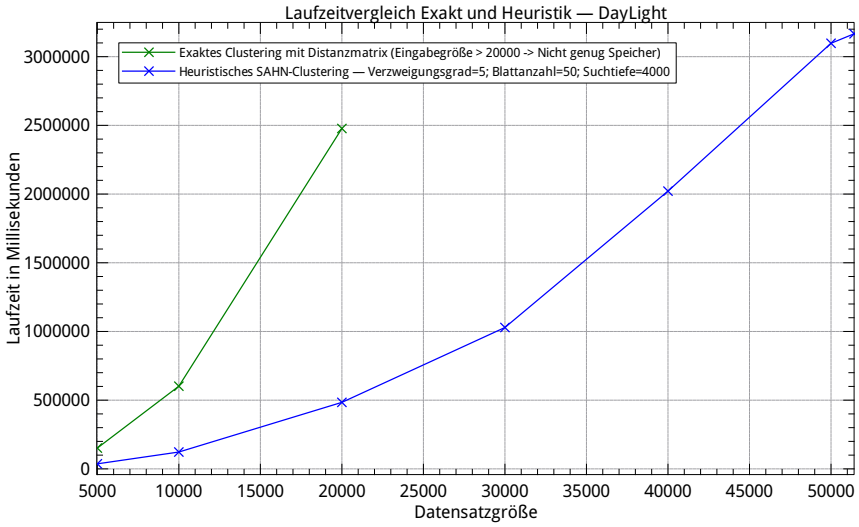



**Abbildung 6.16.:** Laufzeitvergleich; Exakt und heuristisch; DayLight-Datensatz

Die Laufzeitkurve des heuristischen SAHN-Clusterings zeigt mit zunehmender Eingabegröße eine deutliche Abflachung. Zwischen der Datensatzgröße 30000 und 50000 scheint sie sogar sublinear zu verlaufen, was sich nicht mit der theoretischen Analyse deckt. Dieser Effekt kann durch Messungenauigkeiten oder durch das datensatzabhängige Laufzeitverhalten des Generic-Clustering-Algorithmus erklärt werden. Zusammenfassend, kann somit auch in der Praxis eine subquadratische Rechenkomplexität nachgewiesen werden, wenn ein fixer Wert für $s$ gewählt wird.

Im Vergleich mit dem exakten Verfahren wird deutlich, dass die Eingabegröße, für Datensätze mit einer Größe von mehr als 5000 Objekten, bei gleicher Laufzeit mehr als verdoppelt werden kann. Der Speicherbedarf stellt ebenfalls einen erheblichen Unterschied dar: Während 5 GiB Arbeitsspeicher für das exakte Clusterverfahren nicht ausreichen, um 25000 oder mehr Objekte zu clustern, kommt das heuristische SAHN-Clusterverfahren mit weniger als 1 GiB Speicher für 50000 Objekte aus. Der lineare Speicherbedarf (bei konstanter Pivot-Anzahl) ist daher für große Datensätze mindestens genau so wichtig wie die Laufzeitverbesserung.

### 6.5.3. Dimensionalität und Clusterseparation

Dieser Abschnitt beschäftigt sich mit dem Einfluss der Dimensionalität und der Datensatzstruktur auf die Qualität des heuristischen SAHN-Clusterverfahrens.

#### 6.5.3.1. Einfluss der Dimensionalität

In Abschnitt 4.2.3 wurde ausführlich erläutert, welchen Einfluss die Dimensionalität der Daten auf die Effektivität des Pivot-Ansatzes ausübt. Da die intrinsische Dimensionalität in realen Datensätzen nicht automatisch mit der Anzahl der Attribute übereinstimmen muss, wurden für das euklidische Distanzmaß synthetische Datensätze mit gleichverteilten Werten erstellt. Im Folgenden wird der Einfluss der Dimensionalität auf die Qualität (analog zum Abschnitt 6.5.2.1) zunächst ohne Beschränkung der Suchtiefe untersucht.

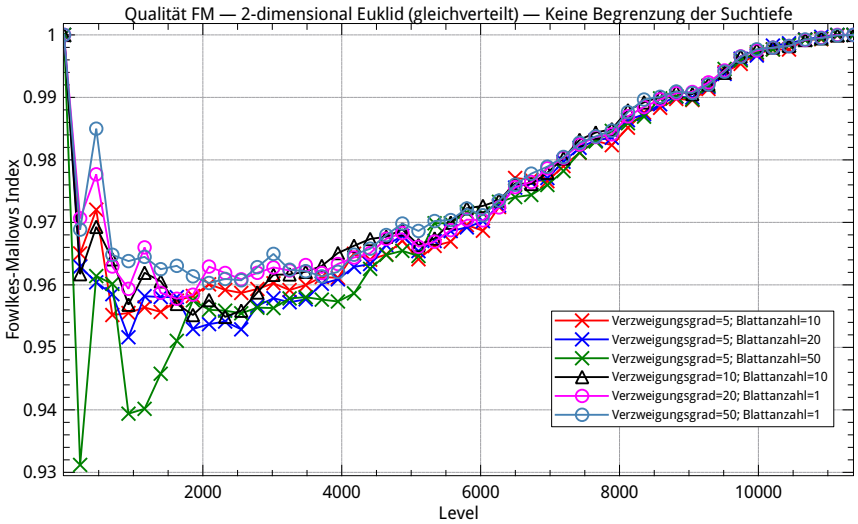



**Abbildung 6.17.:** Einfluss der Dimensionalität auf die Qualität; Euklidisches Distanzmaß; 2-dimensional; Gleichverteilt; Keine Begrenzung der Suchtiefe

Das Clustering des zweidimensionale Datensatzes (vgl. Abbildung 6.17) weist, unabhängig von der Parametereinstellung, eine sehr gute Qualität auf. Dies war aufgrund der

niedrigen Dimensionalität zu erwarten. Interessanterweise zeigt die Parametereinstellung mit $l = 50$ eine leichte Schwäche gegenüber den anderen Parametereinstellungen auf. Diese ist jedoch so gering, dass es sich um einen Zufall handeln kann.

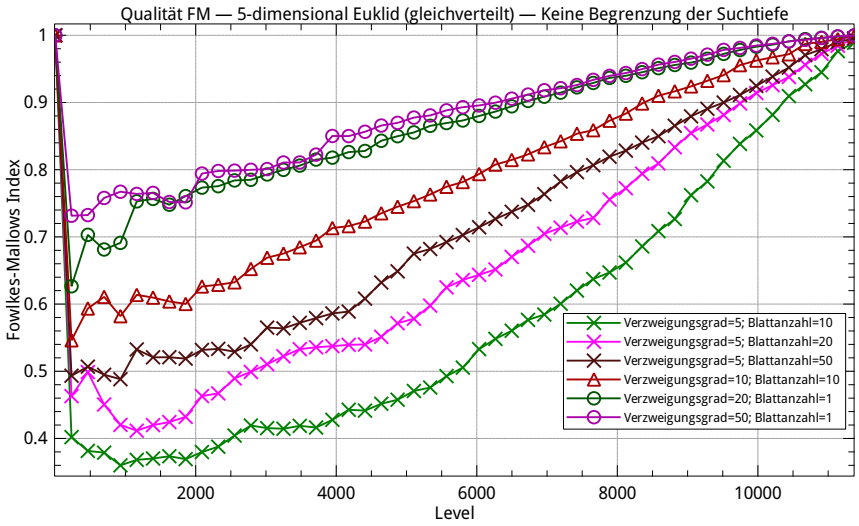



**Abbildung 6.18.:** Einfluss der Dimensionalität auf die Qualität; Euklidisches Distanzmaß; 5-dimensional; Gleichverteilt; Keine Begrenzung der Suchtiefe

Wird die Dimensionalität des Datensatzes auf fünf erhöht (vgl. Abbildung 6.18), zeigt sich ein anderes Bild: Die Qualität aller Clusterings ist deutlich schlechter als im zweidimensionalen Fall und die Pivot-Anzahl beeinflusst die Qualität des Clusterings jetzt wesentlich. Auch die Wahl einer hohen maximalen Blattanzahl $l$ hat einen deutlich positiven Einfluss auf die Qualität. Allein die Parametereinstellungen mit $f = 20$ und $f = 50$ unterscheiden sich nicht wesentlich. Für den fünfdimensionalen Raum scheinen 20 Pivots daher ausreichend zu sein.

Dieses Verhalten spiegelt die theoretische Erkenntnis wieder, dass mit erhöhter Dimensionalität die Anzahl der Pivots steigen muss. Es zeigt leider auch, dass die Dimensionalität nicht unbegrenzt gesteigert werden kann, da mit keiner Parametereinstellung eine annähernd so gute Qualität wie im zweidimensionalen Fall erreicht werden konnte und die Erhöhung der Pivot-Anzahl ab einem gewissen Schwellenwert die Qualität nicht weiter steigert.

#### 6.5.3.2. Einfluss der Clusterseparation

Wird die Suchtiefe der Best-Frontier-Suche beschränkt, fällt die Qualität des Clusterings für den fünfdimensionalen Datensatz, unabhängig von den Parametereinstellungen, auf ein sehr niedriges Niveau (vgl. Abbildung 6.19). Die Qualität aller Clusterings liegt sogar deutlich unter der Qualität, welche für den EState- und den DayLight-Datensatz gemessen

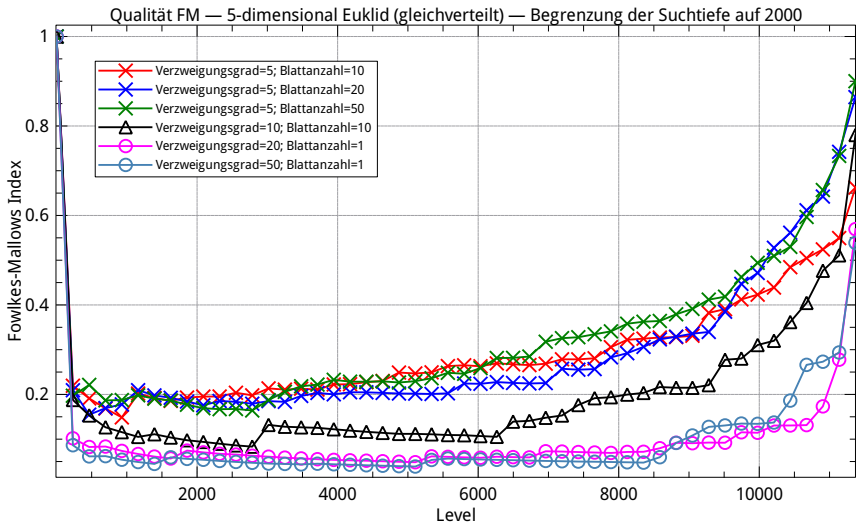



**Abbildung 6.19.:** Einfluss der Dimensionalität auf die Qualität; Euklidisches Distanzmaß; 5-dimensional; Gleichverteilt, Begrenzung der Suchtiefe auf 2000

wurde. Warum ist das der Fall, obwohl die zuvor genannten Datensätze eine intrinsische Dimensionalität aufweisen, welche deutlich über dem Wert fünf liegt?

Die Ursache muss in der Struktur der Daten begründet liegen und eine mögliche Erklärung ist die Folgende: Die euklidischen Testdatensätze sind mit gleichverteilten Werten erstellt worden. Das hat zur Folge, dass keine *echten* Cluster in den Daten existieren und damit alle Clusterings nur eine gleichmäßige Partitionierung des Raumes darstellen. Wo die Grenzen dieser Partitionierung gezogen werden, kann recht willkürlich sein, da die Distanzen nebeneinanderliegender Objekte relativ ähnlich sind und diese in nicht-synthetischen Datensätzen üblicherweise durch Messfehler in der Datenerhebung zusätzlich verrauscht sind. Datensätze mit gleichverteilten Objekten eigenen sich daher prinzipiell schlecht, um mittels Clusteranalyse Eigenschaften der Daten zu extrahieren.

In gleichverteilten Datensätzen kann zusätzlich ein Effekt auftreten, der die Qualität des heuristischen SAHN-Clusterverfahrens negativ beeinflusst. Schon während der Analyse der Qualität des Clustering für die nicht-synthetischen Datensätze wurden kritischen Fehlentscheidungen und Folgefehler beobachtet (vgl. Abschnitt 6.5.2.1). Eine mögliche Erklärung hierfür ist ein Phänomen, welches im Folgenden *Clusterwanderung* genannt wird.

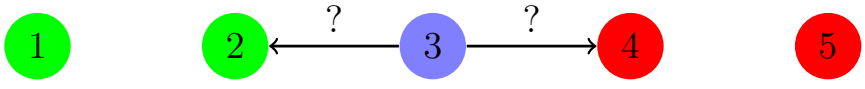



**Abbildung 6.20.:** Clusterwanderung

Abbildung 6.20 zeigt einen eindimensionalen euklidischen Datensatz mit gleichverteilten Attributwerten, in dem Distanzen zu benachbarten Objekten ungefähr gleich groß sind.

Nehmen wir an, dass Cluster 3 mit seinem nächsten Nachbarn verschmolzen werden soll, da die Distanz zwischen Cluster 3 und seinem nächsten Nachbarn der minimalen Distanz im Datensatz entspricht. Da die Distanzunterschiede zu Cluster 2 und 4 nur sehr klein sind, kann im heuristischen Fall leicht der falsche nächste Nachbar ermittelt werden. Wenn beispielsweise Cluster 2 der echte nächste Nachbar ist und Cluster 4 der heuristisch ermittelte, dann hat das zur Folge, dass der Clusterzentroid des verschmolzenen Clusters in die falsche Richtung *wandert*. Dadurch steigt die Wahrscheinlichkeit, dass der Supercluster (ein Cluster, welche aus der Verschmelzung des genannten Clusters entstanden ist) von Cluster 3 und 4 später mit einem noch weiter rechts liegenden Cluster verschmolzen wird und die Objekte aus Cluster 3 und 2 sehr viel später zu einem Cluster verschmolzen werden. Der Clusterzentroid der Supercluster von Cluster 3 entfernt sich also immer weiter von der Position, die er eingenommen hätte, wenn Cluster 3 mit dem linken Nachbarn verschmolzen worden wäre. Eine Korrektur des Fehlers wird immer unwahrscheinlicher, da der Zentroid des Superclusters von Cluster 2 (z. B. durch Verschmelzung mit Cluster 1) weiter nach links wandern kann (durch den vorhergehenden Fehler ebenfalls begünstigt) und die Distanz zwischen den Superclustern von Cluster 3 und 2 damit immer weiter wächst. Die Positionen der Clusterzentroiden werden mit dem weiteren Verlauf des Algorithmus somit immer willkürlicher.

Ein solcher Wanderprozess kann korrigiert werden, falls gut separierte *reale* Cluster in den Daten existieren und die maximale (evtl. fehlerbehaftete) Intra-Cluster-Distanz eines solchen realen Clusters kleiner ist als die Inter-Cluster-Distanzen. In diesem Fall ist garantiert, dass alle Objekte des realen Clusters miteinander verschmolzen werden, bevor Teile von zwei unterschiedlichen realen Clustern verschmolzen werden. Bei guter Clusterseparation sollte das Clustering auf dem entsprechenden Level daher auch eine gute Qualität aufweisen.

Dieser positive, theoretische Effekt ist auch in der Praxis nachzuweisen. Deutlich ist in Abbildung 6.21 zu erkennen, dass für den synthetischen Datensatz mit 32 normalverteilten Clustern (vgl. Abschnitt 6.4.1) das Level 32 eine besonders hohe Qualität aufweist. Für den gleichverteilten Datensatz fällt die Qualität hingegen in den hohen Leveln stark ab. Es ist daher möglich, die realen Cluster eines Datensatzes mit den heuristischen SAHN-Clusterverfahren zu identifizieren, auch wenn andere Teile der Clusterhierarchie evtl. stark vom exakten Clustering abweichen. Dieses Verhalten ist besonders für die Praxistauglichkeit des Verfahrens relevant, denn das Ziel der Clusteranalyse ist das Finden von realen Clustern.

Eine gute Clusterseparation kann auch rekursiv innerhalb von realen Clustern vorhanden sein. Diese Beobachtung ist wichtig, um zu verstehen, warum hierarchische Clusterverfahren einen Mehrwert gegenüber flachen Clusterverfahren bieten.

Die Clusterwanderung beschränkt sich nämlich nicht nur auf heuristische Clusterverfahren. Falls die Daten verrauscht oder fehlerbehaftet sind, produziert ein exaktes Ver-

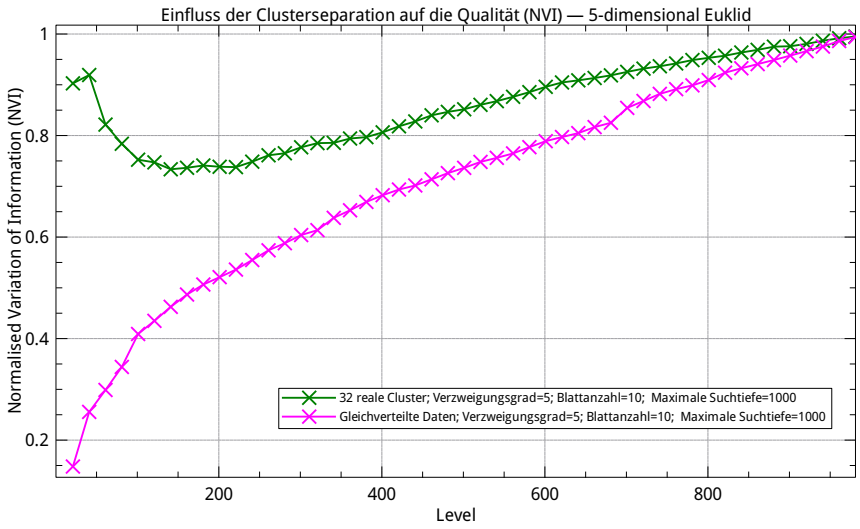



**Abbildung 6.21.:** Einfluss der Clusterseparation auf die Qualität (NVI); 5-dimensional; Euklidisches Distanzmaß

fahren ebenfalls sehr unterschiedliche Clusterings, falls mehrere Messungen des gleichen Realweltereignisses bzw. Pertubationen geclustert werden. Auch hier tritt jedoch der korrigierende Effekt guter Clusterseparation auf. In Analogie zur numerischen Stabilität von Algorithmen, kann festgestellt werden, dass der Rückwärtsfehler sehr klein im Verhältnis zur inversen Qualität des Clusterings ist.

### 6.5.4. Beschränkung der Suchtiefe im Verhältnis zur Eingabegröße

Die Beschränkung der maximalen Suchtiefe $s$ kann zu einer subquadratischen Laufzeit des heuristischen SAHN-Clusterverfahrens führen. Dazu muss die Größe von $s$ sublinear im Verhältnis zur Eingabegröße gewählt werden (vgl. Abschnitt 5.3.3.1). Es bleibt zu überprüfen, ob bei einer sublinearen Wahl von $s$ eine konstante Qualität der Clusterings für unterschiedliche Datensatzgrößen in der Praxis gezeigt werden kann, denn das Verfahren ist unbrauchbar, falls es für sehr große Datensätze nur eine mangelhafte Qualität bei guter Laufzeit ermöglicht.

Die einfachste Möglichkeit den Einfluss der Suchtiefe zu messen besteht darin, den Anteil der korrekten Nächste-Nachbar-Suchen für alle Objekte eines Datensatzes zu zählen und die Suchtiefe zu ermitteln, die notwendig ist, um einen fixen Anteil an korrekten Nächste-Nachbar-Suchen zu erzielen. Anschließend kann das Verhalten im Vergleich zur Eingabegröße mit einem Plot dieser Werte analysiert werden. Dieses Vorgehen würde die Frage nach dem Zusammenhang der Suchtiefe und der Eingabegröße in Bezug auf die Best-Frontier-Suche beantworten, die Folgefehlerproblematik des heuristischen SAHN-Clusterverfahrens aber völlig außer Acht lassen. Daher ist es nicht geeignet, um die ursprüngliche Frage zu beantworten.

Wenn hierarchische Clusterings verglichen werden, existiert eine Folge von Qualitätswerten und es stellt sich die Frage, wie diese zu einem Wert zusammengefasst werden können. Eine einfache und im Folgenden verwendete Methode besteht darin den Durchschnittswert zu berechnen. Da jedoch das höchste und niedrigste Level immer den Qualitätswert 1 aufweist, wird mit dem Durchschnittswert der Vergleich verzerrt. Um diesem Effekt entgegen zu wirken, werden die unteren und oberen 10% der Qualitätswerte nicht in die Durchschnittsberechnung einbezogen.

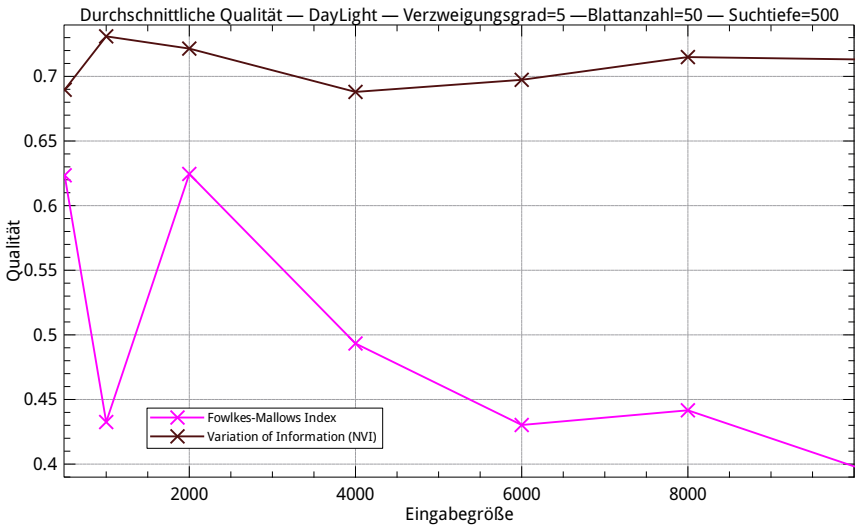



**Abbildung 6.22.:** Durchschnittliche Qualität im Verhältnis zur Eingabegröße; DayLight-Datensatz; Verzweigungsgrad=5; Blattanzahl=50; Suchtiefe=500

In dieser Arbeit werden ausschließlich die Parametereinstellung $f = 5$; $l = 50$ und der DayLight-Datensatz genauer dargestellt, da sich die anderen Datensätze sehr ähnlich verhalten und keine zusätzlichen Erkenntnisse ermöglichen. Abbildung 6.22 zeigt die durchschnittliche Qualität im Verhältnis zur Eingabegröße. Für das NVI-Qualitätsmaß sind die Werte im Trend konstant. Das weist darauf hin, dass die Wahl von $s$ unabhängig von der Eingabegröße ist und ein sehr gutes Laufzeitverhalten ermöglicht. Ein leicht abweichendes Verhalten zeigt der Fowlkes-Mallows-Index. Hier ist ein antiproportionaler Trend zur Eingabegröße zu beobachten.

An dieser Stelle lohnt es sich noch einmal das Verhalten der Qualitätsmaße bei zufälligen Clusterings zu betrachten. Für den Fowlkes-Mallows-Index wurde in Abbildung 6.2 gezeigt, dass der Erwartungswert nicht unabhängig von der Clusteranzahl ist und sich für niedrige Level immer weiter dem Wert 0 annähert. Das hat zur Folge, dass der durchschnittliche Qualitätswert für große Datensätze negativ beeinflusst wird. Das NVI-Qualitätsmaß besitzt diese negative Eigenschaft nicht und sollte daher deutlich aussagekräftiger sein.

Unabhängig von dieser Diskussion ist die zur Erhaltung einer gleichbleibenden Qualität notwendige Steigerung des Parameters $s$ so gering, dass die Granularität der verwendeten Parametereinstellungen zu grob ist, um daraus eine fundierte Aussage über die notwendige Steigerung von $s$ abzuleiten. Eine Steigerung der Suchtiefe um 10% reicht aus, um für einen Datensatz der Größe 10000 eine bessere Qualität zu erreichen, als wenn nur 500 Objekte mit der geringeren maximalen Suchtiefe geclustert werden.

Zusammenfassend lässt sich feststellen, dass die maximale Suchtiefe $s$ wahrscheinlich konstant gewählt werden kann. Sollte aufgrund von Messfehlern oder unbekannten Eigenschaften der Qualitätsmaße dennoch eine Steigerung notwendig sein, ist diese sehr gering. Die Datensatzgröße scheint einen deutlich geringeren Einfluss auf den Parameter $s$ zu haben, als die Struktur der Daten (vgl. Abschnitt 6.5.3.2).

### 6.5.5. Extraktion sinnvoller Parameterkombinationen

Es wurde gezeigt, dass die Wahl der richtigen Parameter von einer Vielzahl unterschiedlicher Einflüsse abhängig ist. Einige Einflüsse haben ihren Ursprung in der Struktur der Daten und sind für den Anwender vermutlich nicht ersichtlich. Zum Beispiel ist die intrinsische Dimensionalität der Daten nicht an der Merkmalsanzahl des euklidischen Distanzmaßes festzumachen und für Fingerprints ist dieser Wert noch weniger offensichtlich. Trotz der unbekannten Größen dieser externen Einflüsse lassen sich aus den Ergebnissen einige Leitsätze ableiten:

1. Eine hohe Blattanzahl $l$ hat einen geringen Einfluss auf die Laufzeit und einen positiven Effekt auf die Qualität, falls die Suchtiefe beschränkt ist. Daher kann diese unabhängig von Datensatz und Distanzmaß immer relativ hoch gewählt werden. Tests mit höherer Blattanzahl wären notwendig um eine konkrete Empfehlung der Blattanzahl abzugeben. Sie sollte aber im Allgemeinen nicht niedriger als 50 liegen.

2. Der Verzweigungsgrad des Pivot-Baumes bzw. die Pivotanzahl pro Baumknoten $f$ hängt im wesentlichen von der Dimensionalität der Daten ab. In den Praxisdatensätzen hat ein Wert zwischen 5 und 10 zu einer guten Qualität bei gleichzeitig guter Laufzeit geführt. Dieser Parameter lässt sich evtl. durch den Anwender festlegen, in dem er die Dimensionalität der Daten mit niedrig ($f \approx 5$), mittel($f \approx 10$) oder hoch ($f \approx 15$) abschätzen kann. Diese sehr einfache Skala würde den Anwender nicht überfordern und hat so wenige Ausprägungen, dass eine vernünftige Einstellung zur Not auch durch Ausprobieren gefunden werden kann. Zudem kann die intrinsische Dimensionalität durch ein Sample der Daten geschätzt werden. In diesem Fall ist die vollautomatische Wahl des Parameters möglich.

3. Die maximale Suchtiefe $s$ ist im wesentlichen von der Struktur der Daten abhängig. In nicht gleichverteilten Datensätzen hat sich jedoch immer ein Wert von unter 4000

als ausreichend erwiesen. Da $s$ nicht oder nur in einem vernachlässigbaren Verhältnis von der Eingabegröße abhängt, kann es statisch festgelegt werden oder über einen Schieberegler zwischen den Extremen *Geschwindigkeit* und *Qualität* auf einem festen Wertebereich konfigurierbar sein. Ein sinnvoller Wertebereich liegt auf Grundlage der hier durchgeführten Messungen zwischen 500 und 10000.

Aufgrund dieser, aus der Praxis abgeleiteten, Regeln lässt sich die Komplexität des Verfahrens für den Anwender ohne spezielle Datamining-Kenntnisse auf ein handhabbares Maß reduzieren.

### 6.5.6. Abschließende Bemerkungen zu den Ergebnissen

Im Laufe der Auswertung der Ergebnisse haben sich einige Schwachstellen des Testdesigns bemerkbar gemacht. Die Qualität des heuristischen SAHN-Clusterverfahrens ist durch die Wahl der Pivots einer gewissen Zufälligkeit unterworfen. Durch Ausreißer bei den Messungen konnten daher häufig keine klaren Zusammenhänge aus den Daten extrahiert werden (s. a. Einfluss der Blattanzahl in Abbildung 6.7). Daher hätten gerade die Qualitätsmessungen gemittelt über mehrere Läufe ausgeführt werden sollen. Durch die mögliche Parallelisierung ist zudem die Laufzeit der Qualitätsmessungen im Vergleich zu den vierfach ausgeführten Performanzmessungen gering gewesen. Es wäre daher sinnvoll gewesen, die Performanzmessungen weniger häufig und die Qualitätsmessungen häufiger auszuführen.

Ein weiterer Verfahrensschwachpunkt war ebenfalls in dieser zufälligen Pivot-Auswahl begründet. Für die verschiedenen Qualitätsmaße wurden jeweils eigene Clusterings berechnet. Das hat zur Folge, dass der Fowlkes-Mallows-Index und das NVI-Qualitätsmaß für die selbe Parametereinstellung und den gleichen Datensatz nur bedingt miteinander vergleichbar sind. Abbildung 6.9 und Abbildung 6.10 zeigen z. B. unterschiedliches Verhalten für die Parametereinstellung $f = 5;\ l = 50$. Es ist unmöglich zu beurteilen, ob die Distanzmaße hier etwas unterschiedliches messen oder der Unterschied durch die zufällige Wahl der Pivots zustande gekommen ist.

In einer Analyse mit einem Profiler wurde festgestellt, dass in etwa $20-25\%$ der Laufzeit durch die Methode *startFrontier()* verursacht wurden. Dies ist auf die Verwendung von Rot-Schwarz-Bäumen in der Klasse *TreeSetDistanceList* anstatt von Hashing zum Auffinden von Objekten in den Distanzlisten zurückzuführen (vgl. Abschnitt 5.4.3). Eine Beschleunigung um etwa $\frac{1}{4}$ der Laufzeit ist daher mit etwas zusätzlichem Implementierungsaufwand möglich.

In Bezug auf die Bewertung der Qualität der Clusterings stellt sich die Frage der Interpretierbarkeit eines Qualitätswertes von z. B. 0,5. Durch die Qualitätsmessungen konnten die Einflüsse verschiedener Faktoren auf die Qualität zwar gut untersucht werden; ob die erreichte Qualität in der Praxis ausreicht, kann jedoch nicht beurteilt werden. Nur ein, an diese Arbeit anschließender, Feldtest könnte diese Fragestellung abschließend beantworten.

# Kapitel 7.

# Zusammenfassung und Ausblick

Um große Datensätze zu clustern, ist die Performanz exakter SAHN-Clusterverfahren unzureichend. Sowohl die Geschwindigkeit, als auch der Speicherbedarf machen das Clustern von mehr als wenigen zehntausend Objekten auf aktueller Hardware unmöglich. Rechenintensive Distanzmaße beeinflussen die Laufzeit stark und schränken daher die Flexibilität in Bezug auf die Wahl eines der Aufgabenstellung entsprechenden, passenden Distanzmaßes ein.

Der Einsatz eines SAHN-Clusterverfahrens im Scaffold Hunter hat die Beschränkung bzgl. der praktischen Anwendbarkeit deutlich gemacht, da zwischen der in der Molekularbiologie üblichen und der durch die Clusteranalyse handhabbaren Datensatzgröße erhebliche Differenzen bestehen. Die Praxistauglichkeit teilweise deutlich schnellerer, nichthierarchischer Clusterverfahren ist durch die Arbeitsweise des Anwenders jedoch eingeschränkt. Daher wurden Möglichkeiten zur Beschleunigung hierarchischer Clusterverfahren untersucht.

Um die intuitive Anpassbarkeit des Clusterverfahrens mittels Wahl eines passenden Distanzmaßes nicht zu stark einzuschränken, mussten viele mögliche Beschleunigungen (z. B. gitterbasierte Clusterverfahren, Indizierung der Daten) verworfen werden, da sie ausschließlich für das euklidische Distanzmaß anwendbar sind. Datenverdichtung und der Pivot-Ansatz haben sich als mögliche Ansätze zum Beschleunigen der Clusterverfahren herausgestellt.

Der Pivot-Ansatz nutzt die Dreiecksungleichung metrischer Distanzmaße und wurde aufgrund des im Vergleich zu den Datenverdichtungsverfahren höheren Informationsgehalts des resultierenden Clusterings weiter verfolgt. Durch den Pivot-Ansatz können Distanzkalkulationen eingespart und heuristische Distanzabschätzungen berechnet werden. Mit Hilfe der suchtiefenbeschränkten Best-Frontier-Suche ist es zudem möglich, die Performanz von Nächste-Nachbar-Suchen asymptotisch zu beschleunigen. Für die Best-Frontier-Suche wurden Modifikationen diskutiert und umgesetzt, welche zu einer höheren Qualität der Nächste-Nachbar-Suchen führen. Um die Effektivität der Pivots für Nächste-Nachbar-Suchen weiter zu erhöhen und dadurch auch die Geschwindigkeit des Verfahrens zu verbessern, wurde eine Pivot-Baum-Datenstruktur eingesetzt, welche es ermög-

licht eine Teilmenge der Pivots nur für lokale Distanzabschätzungen zu verwenden. Die ursprüngliche Pivot-Baum-Datenstruktur wurde angepasst, um den Anforderungen des SAHN-Clusterverfahrens zu entsprechen und es wurden Methoden spezifiziert, mit denen Cluster aus der Datenstruktur gelöscht sowie in die Datenstruktur eingefügt werden können.

Der im Scaffold Hunter bisher eingesetzte NNChain-Clustering-Algorithmus ist für heuristische Nächste-Nachbar-Suchen nicht geeignet. Daher wurde ein modifizierter Generic-Clustering-Algorithmus verwendet, welcher im *Best-Case* eine subquadratische Laufzeit besitzt, falls eine sublineare Nächste-Nachbar-Suche eingesetzt wird.

Durch die Kombination des modifizierten Generic-Clustering-Algorithmus mit den angepassten Varianten der Best-Frontier-Suche und des Pivot-Baumes konnte ein neues heuristisches SAHN-Clusterverfahren entwickelt werden, welches anschließend implementiert und in der Praxis getestet wurde. Die bestehenden Programmstrukturen wurden analysiert und modularisiert, um verschiedene Clusterverfahren, Nächste-Nachbar-Suchstrategien, Distanzmaße und Linkage-Verfahren miteinander kombinieren zu können.

Um die Qualität des heuristischen SAHN-Clusterverfahrens beurteilen zu können, wurden für die Evaluation verschiedene Qualitätsmaße vorgestellt und ihr Verhalten diskutiert, um die Qualitätsmessungen interpretieren zu können. Ein Evaluationsframework wurde implementiert, um das Testdesign flexibel gestalten, verschiedene Parametereinstellungen systematisch miteinander vergleichen und die Messergebnisse für eine spätere Auswertung abzuspeichern zu können.

Die Testergebnisse haben gezeigt, dass das heuristische SAHN-Clusterverfahren die Laufzeit erheblich verbessern kann und in der Praxis eine subquadratische Laufzeit ermöglicht. Auch der lineare Speicherbedarf in Bezug auf die Eingabegröße trägt dazu bei, dass die handhabbare Datensatzgröße für die Clusteranalyse im Scaffold Hunter um ein vielfaches gesteigert werden konnte. Rechenintensive Distanzmaße haben einen deutlich geringeren Einfluss auf die Gesamtlaufzeit des Algorithmus, da in Bezug auf die Eingabegröße nur noch linear viele Distanzberechnungen notwendig sind.

Die Qualitätsmessungen haben gezeigt, dass die Qualität des Clusterings unabhängig von der Datensatzgröße ein gleichbleibendes Niveau erreicht. Sie ist jedoch nicht unabhängig von Struktur und Dimensionalität der Daten, denn eine hohe Dimensionalität und gleichverteilte Merkmale wirken sich negativ aus. Für die besonders praxisrelevanten Datensätze unter Verwendung von DayLight- und EState-Fingerprint konnte eine solide Qualität nachgewiesen werden. Es bleibt in einem Feldtest zu überprüfen, ob diese Qualität ausreichend ist, da die Qualitätswerte in Bezug auf die Praxistauglichkeit nur bedingt aussagekräftig sind.

Während der Umsetzung wurde aus Zeitgründen nicht das volle Optimierungspotenzial ausgeschöpft. Daher sind im Folgenden einige Gebiete aufgeführt, für die eine weitere Vertiefung der Forschung oder eine verbesserte Implementierung lohnenswert erscheint.

Die Implementierung der Distanzlisten ist nicht optimal und benötigt beim Starten einer neuen Best-Frontier-Suche logarithmische viel Zeit für das Auffinden einzelner Objekte. Durch die Verwendung von Hashing ist es möglich diese Laufzeit auf konstante Zeit zu reduzieren und damit schätzungsweise 25% der Laufzeit zu sparen.

Eine bisher ungeklärte Fragestellung ist, ob die Effektivität der Pivots durch eine zum zufälligen Sampling alternative Auswahlstrategie gesteigert werden kann. Hierzu gibt es in der wissenschaftlichen Literatur vielversprechende Ansätze. Die Implementierung besitzt bereits ein Interface für unterschiedliche Auswahlstrategien und kann daher einfach erweitert werden.

Die Reduzierung der Tiefe des Pivot-Baumes durch die Clusterverschmelzungen ist vermeidbar, falls eine alternative Einfügeoperation für den dynamischen Pivot-Baum verwendet wird. Aus Zeitgründen wurde diese alternative jedoch nicht umgesetzt und daher besteht auch auf diesem Gebiet noch Forschungsbedarf. Es muss untersucht werden, ob die alternative Einfügeoperation zu einer verbesserten Qualität des Clustering führt und wie sie sich auf die Gesamtlaufzeit des Algorithmus auswirkt.

Durch die Verwendung der heuristischen Nächste-Nachbar-Suche kommt es im Dendrogramm des Clustering häufig zu sogenannten Reversals. Diese behindern die Leserlichkeit des Ergebnisses und sollte daher möglichst eliminiert werden. Hierfür kann die Höhe eines Dendrogrammknoten in einem Nachverarbeitungsschritt korrigiert werden, indem aus dem Maximum seiner eigenen Höhe und den Höhen seiner Kinder die angepasste Höhe berechnet wird. Es gibt noch eine Vielzahl weiterer Möglichkeiten, die Darstellung des Dendrogramms insbesondere in Bezug auf sehr große Datensätze zu verbessern. Die Optimierung der Darstellungskomponente bleibt damit zukünftigen Arbeiten vorbehalten.

# Anhang A.

# Abkürzungsverzeichnis

| Abkürzung | Langform |
|---|---|
| DBSCAN | Density-Based Spatial Clustering of Applications with Noise |
| LCA | Lowest Common Ancestor |
| MCS | Maximal Common Subgraph (Größter Gemeinsamer Subgraph) |
| MST | Minimal Spanning Tree (Minimaler Spannbaum) |
| MVC | Model-View-Controller (Entwurfsmuster) |
| NN | Nächster Nachbar |
| OPTICS | Ordering Points To Identify the Clustering Structure |
| RNN | Reziproker nächster Nachbar |
| SAHN | Sequential Agglomerative Hierarchical Non-Overlapping (Clustering) |
| VI | Variation of Information |

# Abbildungsverzeichnis

# Algorithmenverzeichnis

# Literaturverzeichnis


[ABK+99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel u. a.: „OPTICS: ordering points to identify the clustering structure“. In: *SIGMOD Rec.* 28.2 (Juni 1999), S. 49–60. ISSN: 0163-5808. DOI: 10.1145/304181.304187.

[BF87] Jon Bentley und Bob Floyd: „Programming pearls: a sample of brilliance“. In: *Commun. ACM* 30.9 (1987), S. 754–757. ISSN: 0001-0782. DOI: 10.1145/30401.315746. URL: http://doi.acm.org/10.1145/30401.315746.

[BKK+01] Markus M. Breunig, Hans peter Kriegel, Peer Kröger u. a.: „Data bubbles: Quality preserving performance boosting for hierarchical clustering“. In: *IN ACM SIGMOD CONFERENCE*. 2001, S. 79–90.

[BKS+90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider u. a.: „The R*-tree: an efficient and robust access method for points and rectangles“. In: *SIGMOD Rec.* 19.2 (1990), S. 322–331. ISSN: 0163-5808. DOI: 10.1145/93605.98741.

[BNC03] Benjamin Bustos, Gonzalo Navarro und Edgar Chavez: „Pivot selection techniques for proximity searching in metric spaces“. In: *Pattern Recognition Letters* (2003).

[Bat88] V. Batagelj: „Generalized Ward and Related Clustering Problems“. In: *Classification and Related Methods of Data Analysis* (1988). Hrsg. von H. H. Bock, S. 67–74.

[CE04] Jean Cardinal und David Eppstein: „Lazy Algorithms for Dynamic Closest Pair with Arbitary Distance Measures“. In: *ALENEX/ANALC*. 2004, S. 112–119.

[CN01] Edgar Chavez und Gonzalo Navarro: „A Probabilistic Spell for the Curse of Dimensionality“. In: *In ALENEX'01, LNCS 2153*. Springer, 2001, S. 147–160.

[DB03] Geoff M. Downs und John M. Barnard: „Clustering Methods and Their Uses in Computational Chemistry“. In: *Reviews in Computational Chemistry*. John Wiley & Sons, Inc., 2003, S. 1–40. ISBN: 9780471433514. DOI: 10.1002/0471433519.ch1.

[DGS+03] Vlastislav Dohnal, Claudio Gennaro, Pasquale Savino u. a.: „D-Index: Distance Searching Index for Metric Data Sets“. In: *Multimedia Tools and Applications* 21 (1 2003), S. 9–33. ISSN: 1380-7501. URL: http://dx.doi.org/10.1023/A:1025026030880.

[DLX01] Manoranjan Dash, Huan Liu und Xiaowei Xu: „'1 + 1 > 2': Merging Distance and Density Based Clustering“. In: *In Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on*. 2001, S. 32–39.

[EKS+96] Martin Ester, Hans peter Kriegel, Jörg S u. a.: „A density-based algorithm for discovering clusters in large spatial databases with noise“. In: AAAI Press, 1996, S. 226–231.

[Elk03] Charles Elkan: „Using the Triangle Inequality to Accelerate k-Means“. In: *ICML'03*. 2003, S. 147–153.

[Epp95] D. Eppstein: „Dynamic Euclidean minimum spanning trees and extrema of binary functions“. In: *Discrete and Computational Geometry* 13.1 (Dez. 1995), S. 111–122. DOI: 10.1007/BF02574030.

[Epp98] David Eppstein: „Fast hierarchical clustering and other applications of dynamic closest pairs“. In: *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*. SODA '98. San Francisco, California, United States: Society for Industrial und Applied Mathematics, 1998, S. 619–628. ISBN: 0-89871-410-9.

[FM83] E. B. Fowlkes und C. L. Mallows: „A Method for Comparing Two Hierarchical Clusterings“. In: *Journal of the American Statistical Association* 78.383 (1983), S. 553–569. ISSN: 01621459. DOI: 10.2307/2288117.

[FVB02] Michael A Fligner, Joseph S Verducci und Paul E Blower: „A Modification of the Jaccard–Tanimoto Similarity Index for Diverse Selection of Chemical Compounds Using Binary Strings“. In: *Technometrics* 44.2 (2002), S. 110–119. DOI: 10.1198/004017002317375064.

[GL86] J.C. Gower und P. Legendre: „Metric and Euclidean properties of dissimilarity coefficients“. In: *Journal of classification* 3.1 (1986), S. 5–48. ISSN: 0176-4268.

[GRS98] Sudipto Guha, Rajeev Rastogi und Kyuseok Shim: „CURE: an efficient clustering algorithm for large databases“. In: *SIGMOD Rec.* 27.2 (Juni 1998), S. 73–84. ISSN: 0163-5808. DOI: 10.1145/276305.276312.

[Gut84] Antonin Guttman: „R-trees: A Dynamic Index Structure for Spatial Searching“. In: *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 1984, S. 47–57.

[HHK98] Alexander Hinneburg, Er Hinneburg und Daniel A. Keim: „An Efficient Approach to Clustering in Large Multimedia Databases with Noise“. In: AAAI Press, 1998, S. 58–65.

[HK99] Alexander Hinneburg und Daniel A. Keim: „Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering“. In: Morgan Kaufmann, 1999, S. 506–517.

[IM98] Piotr Indyk und Rajeev Motwani: „Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality“. In: *STOC*. ACM, 1998, S. 604–613. ISBN: 0-89791-962-9.

[KHI+07] Koga, Hisashi, Ishibashi u. a.: „Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing“. In: *Knowledge and Information Systems* 12.1 (2007), S. 25–53. ISSN: 0219-1377. DOI: 10.1007/s10115-006-0027-5.

[KHK99] G. Karypis, Eui-Hong Han und V. Kumar: „Chameleon: hierarchical clustering using dynamic modeling“. In: *Computer* 32.8 (1999), S. 68–75. ISSN: 0018-9162. DOI: 10.1109/2.781637.

[Kel75] J.L. Kelley: *General Topology*. Graduate Texts in Mathematics. Springer-Verlag, 1975. ISBN: 9780387901251.

[LW67] G. N. Lance und W. T. Williams: „A general theory of classificatory sorting strategies 1. Hierarchical systems“. In: *The Computer Journal* 9.4 (1967), S. 373–380. DOI: doi:10.1093/comjnl/9.4.373.

[Lip99] Alan Lipkus: „A proof of the triangle inequality for the Tanimoto distance“. In: *Journal of Mathematical Chemistry* 26 (1 1999), S. 263–265. ISSN: 0259-9791. DOI: 10.1023/A:1019154432472.

[MC12] Fionn Murtagh und Pedro Contreras: „Algorithms for hierarchical clustering: an overview.“ In: *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery* 2.1 (2012), S. 86–97.

[Mei07] Marina Meilă: „Comparing clusterings — an information based distance“. In: *J. Multivar. Anal.* 98.5 (2007), S. 873–895. ISSN: 0047-259X. DOI: 10.1016/j.jmva.2006.11.013.

[Mur85] F. Murtagh: „Multidimensional Clustering Algorithms“. In: *COMPSTAT Lectures 4*. Wuerzburg: Physica-Verlag, 1985. Kap. 4.

[Mö98] Hanspeter Mössenböck: *Objektorientierte Programmierung in Oberon-2 (3. Aufl.)* Springer, 1998, S. I–XVII, 1–327. ISBN: 978-3-540-64649-5.

[Mü11] Daniel Müllner: „Modern hierarchical, agglomerative clustering algorithms“. arXiv:1109.2378v1. 2011.

[PHB+10] Bidyut Kr. Patra, Neminath Hubballi, Santosh Biswas u. a.: „Distance based fast hierarchical clustering method for large datasets“. In: *Proceedings of the 7th international conference on Rough sets and current trends in computing*. RSCTC'10. Springer-Verlag, 2010, S. 50–59. ISBN: 3-642-13528-5, 978-3-642-13528-6.

[Ran71] W.M. Rand: „Objective criteria for the evaluation of clustering methods“. In: *Journal of the American Statistical Association* 66.336 (1971), S. 846–850.

[Roh73] F. James Rohlf: „Hierarchical clustering using the minimum spanning tree“. In: *Comput. Journal* 16 (1973), 93–95.

[SCZ00] Gholamhosein Sheikholeslami, Surojit Chatterjee und Aidong Zhang: „WaveCluster: a wavelet-based clustering approach for spatial data in very large databases“. In: *The VLDB Journal* 8.3-4 (2000), S. 289–304. ISSN: 1066-8888. DOI: 10.1007/s007780050009.

[SER+07] Ansgar Schuffenhauer, Peter Ertl, Silvio Roggo u. a.: „The Scaffold Tree - Visualization of the Scaffold Universe by Hierarchical Scaffold Classification“. In: *Journal of Chemical Information and Modeling* 47.1 (2007), S. 47–58. DOI: 10.1021/ci600338x.

[SHR97] Anja Struyf, Mia Hubert und Peter Rousseeuw: „Clustering in an Object-Oriented Environment“. In: *Journal of Statistical Software* 1.4 (Feb. 1997), S. 1–30. ISSN: 1548-7660.

[SYM10] D.S. Starnes, D. Yates und D. Moore: *The Practice of Statistics*. W. H. Freeman, 2010. ISBN: 9781429245593.

[Sch07] Satu Elisa Schaeffer: „Graph clustering“. In: *Computer Science Review* 1.1 (2007), S. 27 –64. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2007.05.001.

[Sch96] E. Schikuta: „Grid-clustering: an efficient hierarchical clustering method for very large data sets“. In: *Pattern Recognition, 1996., Proceedings of the 13th International Conference on* 2 (Aug. 1996), 101–105 vol.2. DOI: 10.1109/ICPR.1996.546732.

[VEB10] Nguyen Xuan Vinh, Julien Epps und James Bailey: „Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance“. In: *J. Mach. Learn. Res.* 11 (2010), S. 2837–2854. ISSN: 1532-4435.

[WKR+09] Stefan Wetzel, Karsten Klein, Steffen Renner u. a.: „Interactive exploration of chemical space with Scaffold Hunter“. In: *Nature Chemical Biology* 5.8 (Juni 2009), S. 581–583. ISSN: 1552-4450. DOI: 10.1038/nchembio.187.

[WYM97] Wei Wang, Jiong Yang und Richard R. Muntz: „STING: A Statistical Information Grid Approach to Spatial Data Mining“. In: *Proceedings of the 23rd International Conference on Very Large Data Bases*. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, S. 186–195. ISBN: 1-55860-470-7.

[XEK+98] Xiaowei Xu, Martin Ester, Hans-Peter Kriegel u. a.: „A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases“. In: *Proceedings of the Fourteenth International Conference on Data Engineering*. ICDE '98. Washington, DC, USA: IEEE Computer Society, 1998, S. 324–331. ISBN: 0-8186-8289-2.

[ZRL96] Tian Zhang, Raghu Ramakrishnan und Miron Livny: „BIRCH: an efficient data clustering method for very large databases“. In: *SIGMOD Rec.* 25.2 (1996), S. 103–114. ISSN: 0163-5808. DOI: 10.1145/235968.233324.

[ZS03] Jianjun Zhou und Jörg Sander: „Data Bubbles for Non-Vector Data: Speeding-up Hierarchical Clustering in Arbitrary Metric Spaces“. In: *Proceedings of the 29th international conference on Very large data bases - Volume 29*. VLDB '03. Berlin, Germany: VLDB Endowment, 2003, S. 452–463. ISBN: 0-12-722442-4.

[ZS06] Jianjun Zhou und Joerg Sander: „Speedup Clustering with Hierarchical Ranking“. In: *Proceedings of the Sixth International Conference on Data Mining*. ICDM '06. Washington, DC, USA: IEEE Computer Society, 2006, S. 1205–1210. ISBN: 0-7695-2701-9. DOI: 10.1109/ICDM.2006.151.

[Zho09] Jianjun Zhou: „Efficiently Searching and Mining Biological Sequence and Structure Data“. Diss. University of Alberta, 2009.

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

Dortmund, den 21. September 2012

Till Schäfer