Diplomarbeit

Effiziente Lokalisierung von Produktmodellen in dezentralen Umgebungen durch inhaltsbasierte Verteilungsstrategien

eingereicht bei

Institut für Informatik
Abteilung für Wirtschaftsinformatik
Fakultät für Mathematik/Informatik und Maschinenbau
Technische Universität Clausthal

von

Malte Aschermann
Tannenhöhe 11
38678 Clausthal-Zellerfeld
Studienrichtung: Wirtschaftsinformatik

Datum: 29. August 2012

Erstgutachter: Prof. Dr. Jörg P. Müller
Zweitgutachter: Prof. Dr. Niels Pinkwart
Betreuer: Dipl.-Wirt.-Inf. Stefan Kehl

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Clausthal-Zellerfeld, 29. August 2012

Malte Aschermann

Erklärung zur Auslage und Einsichtnahme

Hiermit erkläre ich mich damit einverstanden, dass meine Diplomarbeit in der Instituts- und/oder Universitätsbibliothek ausgelegt und zur Einsichtnahme aufbewahrt werden darf.

Clausthal-Zellerfeld, 29. August 2012

Malte Aschermann

Abstract

Eine Möglichkeit zur effizienten Verwaltung von Produktmodellen stellen Datenmanagementsysteme auf der Basis von Peer-to-Peer-Overlay-Netzwerken dar. Hierbei werden Ressourcen durch den Einsatz von Hash-Funktionen eindeutig identifiziert und sind somit effizient lokalisierbar. Dies ist jedoch nur dann sinnvoll, wenn die Bezeichner der gewünschten Ressourcen bekannt sind. Soll hingegen inhaltsbezogen nach ähnlichen Daten gesucht werden, so ist ein solcher Ansatz nicht zielführend. Für eine inhaltsbasierte Lokalisierung von ähnlichen Daten stellt sich daher die Frage, wie zu einer vorgegebenen Ressourcenspezifikation alle ähnlichen Ressourcen gefunden werden können, ohne bei jeder Suchanfrage sämtliche Datenobjekte abgleichen zu müssen. Dieses Problem wird gerade bei einer sehr hohen Anzahl an Teilnehmern und entsprechend vielen Ressourcen relevant. Zudem können in Netzwerken, in denen sich (geographisch oder durch unternehmerische Planung) abgekapselte Gemeinschaften von Teilnehmern (Pools) bilden, die Übertragungskosten zwischen den Pools signifikant werden, wenn Ressourcen lokalisiert werden sollen. Im Rahmen dieser Arbeit sollen Möglichkeiten aufgezeigt werden, wie Ressourcen anhand paarweiser Ähnlichkeiten gruppiert und in verteilten Systemen organisiert werden können, so dass irrelevante Gruppierungen schnell erkannt und der Suchraum vorab minimiert werden kann. Um die oben beschriebenen Funktionen bieten zu können, wird eine abstrakte P2P-Architektur erweitert. Der daraus resultierende Vorteil liegt in der Unabhängigkeit von den zugrundeliegenden P2P-Systemen. Zu diesem Zweck werden in dieser Arbeit Clustering-Verfahren unter dem Aspekt untersucht, ob diese für das gegebene Szenario geeignet sind; weiterhin werden Möglichkeiten entwickelt, wie eine Verwaltung von ausprägungsähnlichen Bauteilen durch lokalsensitive Hash-Funktionen denkbar ist. Abschließend werden die untersuchten Verfahren sowie die aus der Problemstellung resultierenden Datenmodelle hinsichtlich ihrer Eignung als effiziente Lösungsansätze miteinander verglichen und bewertet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

1 Einleitung

In produzierenden Unternehmen werden zur Verwaltung von Produktmodellen – unter anderem bei kollaborativen Entwicklungen – Systeme zum Produktdatenmanagement (PDM) eingesetzt. Diese umfassen unter anderem den Aufbau von Produkten, in Form von Baugruppen bis hin zu Bauteilen, sowie hierfür relevante Umfelddaten, wie zum Beispiel Computer-Aided Design (CAD) Zeichnungen. Neben dem zentralisierten Einsatz von PDM-Systemen ist auch ein verteilter Ansatz denkbar, der im Kontext einer *dezentralen kollaborativen Produktentwicklung* von Stiefel [Sti10] untersucht wurde. Aus der Art der Verwaltung von Produktmodellen in verteilten Umgebungen folgt die Motivation die im folgenden Abschnitt erläutert wird.

1.1 Motivation und Problemstellung

Eine Möglichkeit zur effizienten Verwaltung von Produktmodellen stellen Datenmanagementsysteme auf der Basis von Peer-to-Peer (P2P) Overlay-Netzwerken dar, wie sie bereits von [Sti10] untersucht wurden.

In solchen Netzwerken werden Ressourcen durch den Einsatz von Hash-Funktionen eindeutig identifiziert und sind somit effizient lokalisierbar. Dies ist jedoch nur dann sinnvoll, wenn die Bezeichner der gewünschten Ressourcen bekannt sind. Dabei sei zu beachten, dass die alleinige Kenntnis eines Bezeichners (oder auch Hash-Wertes) nicht ausreicht, um direkt auf den Inhalt der jeweiligen Ressource zu schließen.

Es ist allerdings auch denkbar, dass eine Referenzspezifikation zu gesuchten Baugruppen nur abstrakt bekannt ist oder schlicht nach Ressourcen gesucht werden soll, die ähnlich zu einer gegebenen sind. Um ähnliche Ressourcen zu lokalisieren oder auch vorab nicht relevante ausschließen zu können, ist ein solcher hashbasierter Ansatz nicht zielführend. Für eine inhaltsbasierte Lokalisierung von Daten stellt sich daher die Frage, wie zu einer vorgegebenen Ressourcenspezifikation ähnliche Ressourcen gefunden werden können, ohne bei jeder Suchanfrage sämtliche Datenobjekte abgleichen zu müssen. Eine solche Suche wird in Abbildung 1.1.1 verdeutlicht, in der ein Teilnehmer, bei der Suche nach ähnlichen Baugruppen eines bestimmten Typs, alle verfügbaren Ressourcen abgleichen müsste, um die gewünschten zu erhalten.

Dieses Problem wird gerade bei einer sehr hohen Anzahl an Teilnehmern und entsprechend vielen Ressourcen relevant. Handelt es sich zudem um Netzwerke, in denen sich beispielsweise Teilnehmer geographisch bedingt in Pools zusammenfinden, so können Übertragungskosten zwischen Teilnehmern in un-

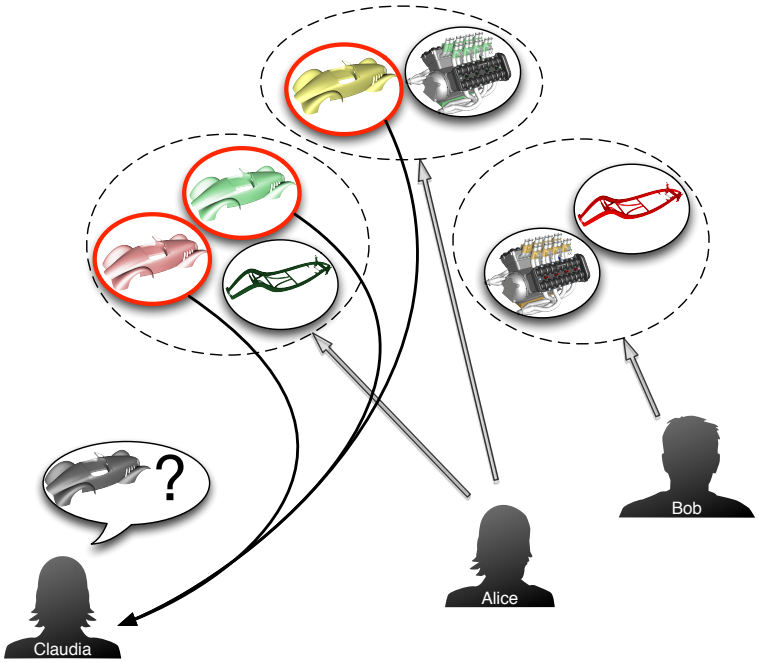


Abbildung 1.1.1: Verteilte Ressourcenverwaltung zwischen Kollaborationspartnern mit Lokalisierung ähnlicher Baugruppen (Baugruppen aus [Sie12b])

terschiedlichen Pools signifikant werden. Siehe hierzu die Abbildung 1.1.2, in der ein Anwendungsszenario zur verteilten Baugruppenverwaltung illustriert wird.

Im Rahmen dieser Arbeit sollen daher Möglichkeiten aufgezeigt werden, wie Ressourcen anhand paarweiser Ähnlichkeiten gruppiert und in verteilten Systemen organisiert werden können, so dass irrelevante Gruppierungen schnell erkannt und der Suchraum vorab minimiert werden kann.

Um die beschriebenen Funktionalitäten realisieren zu können, soll eine abstrakte P2P-Architektur erweitert werden, um eine Unabhängigkeit von der tatsächlichen Implementierung zu erreichen.

Darüber hinaus sollen zur Identifikation ähnlicher Ressourcen Clustering-Verfahren unter dem Aspekt untersucht werden, ob diese für das gegebene Szenario geeignet sind; weiterhin werden Möglichkeiten entwickelt, wie eine Verwaltung von ausprägungsähnlichen Bauteilen durch lokal-sensitive Hashfunktionen denkbar ist.

1.2 Vorgehen

Es werden zunächst in Kapitel 2 die Grundlagen erläutert, die für den weiteren Verlauf dieser Arbeit notwendig sind. Dabei wird auf die allgemeine Struktur von Netzgraphen und Daten eingegangen sowie beschrieben wie in solchen

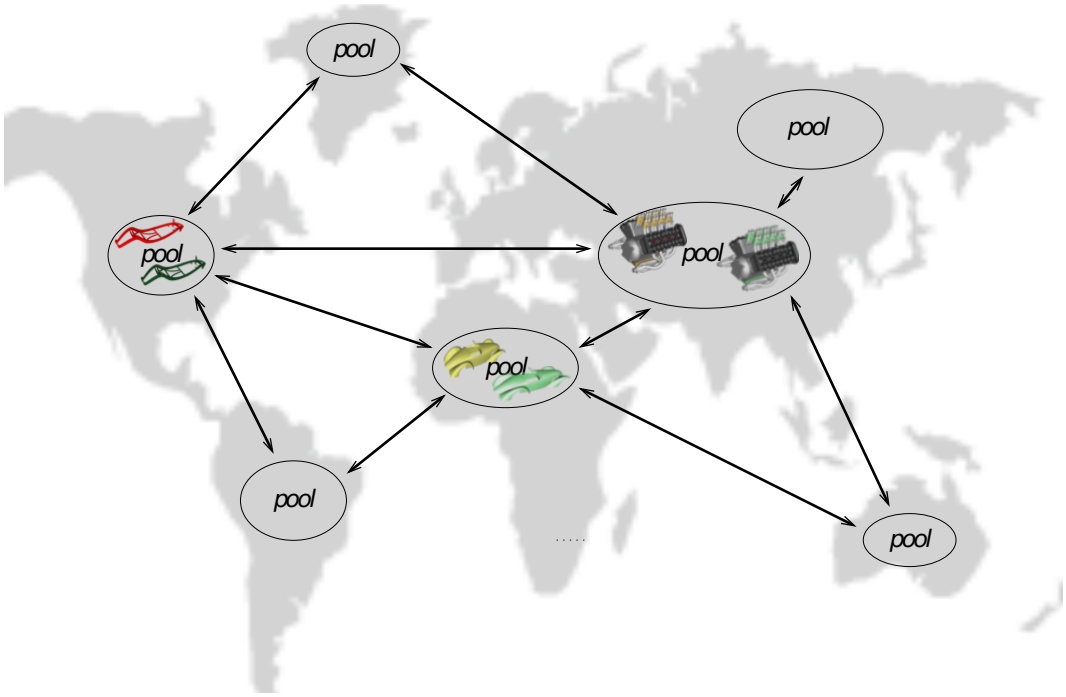


Abbildung 1.1.2: Verteilte Ressourcenverwaltung in Netzwerk-Pools (Netzgraph aus [Kub+00, S.2], Baugruppen aus [Sie12b])

Strukturen Abstände und Ähnlichkeiten bestimmt werden können. Des Weiteren werden die Grundprinzipien von Hash-Funktionen vorgestellt. Ein wichtiger Abschnitt befasst sich zudem mit der Transformation reellwertiger Daten in Graphen. Die Relevanz zur Überführung von Datenpunkten in Graphstrukturen zeigt sich in der anschließenden Vorstellung unterschiedlicher Clustering-Verfahren, die, entsprechend der Problemstellung dieser Arbeit, näher untersucht werden sollen.

Anschließend soll in Kapitel 3 auf Ansätze und Systeme eingegangen werden, die Probleme behandeln, die mit dem Untersuchungsgegenstand dieser Arbeit verwandt sind.

Kapitel 4 und Kapitel 5 schließlich umfassen den Hauptteil dieser Arbeit. In Kapitel 4 wird die *Ähnlichkeitsbasierte Verwaltung von Produktmodellen* thematisiert. Darin werden bestehende Techniken zur Verteilung von Produktdaten in Netzwerken diskutiert und darauf aufsetzende Datenmodelle spezifiziert, die geeignet sind, eine ähnlichkeitsbasierte Lokalisierung zu ermöglichen. Im Anschluss daran werden die entworfenen Modelle bezüglich der Anforderungen, die aus der Problemstellung dieser Arbeit folgen, evaluiert. Kapitel 5 schließlich behandelt die eingangs genannte *verteilte Netzwerkstruktur mit Pools* (siehe Abbildung 1.1.2). Hierbei werden die Lokalisierungsabläufe bezüglich des gesamten Netzwerkes und der Pools spezifiziert. Zudem findet eine Abstraktion

von Netzwerken auf Netzgraphen statt, um Gemeinschaften und Repräsentanten identifizieren zu können.

Abschließend wird ein Gesamtfazit gezogen, inwiefern die durchgeführten Untersuchungen und Ergebnisse dieser Arbeit die Probleme und Fragestellungen lösen konnten.

Noch offene Punkte und weitere Bereiche in denen tiefgehendere Untersuchungen denkbar sind, werden im Ausblick angesprochen.

2 Hintergrund

Die Grundlagen, die im weiteren Verlauf dieser Arbeit wichtig sind, sollen in diesem Kapitel vermittelt werden. In den nachfolgenden Abschnitten sollen grundsätzliche Verfahren und Definitionen vorgestellt werden, die für eine Beschreibung von Netzgraphen, Datenobjekten, sowie der Identifikation von Clustern notwendig sind. Hierbei wird schwerpunktmäßig auf die Berechnung von Abständen und Ähnlichkeiten zwischen Daten und in Graphen, sowie die Überführung von Datenpunkten in Graphenstrukturen eingegangen. Die Relevanz dieser Verfahren zeigt sich im weiteren Verlauf, wenn Clustering-Algorithmen vorgestellt werden, um ähnliche Daten bzw. Gemeinschaften in Graphen zu identifizieren.

2.1 Datenformate

Soll bei der Verwaltung von strukturierten Daten auch die Semantik beachtet werden, so ist von entscheidender Bedeutung, in welchem Format sich diese Daten befinden. Hierbei soll vereinfachend davon ausgegangen werden, dass Datenobjekte, die zu einer bestimmten Datenmenge X gehören, unter sich homogen sind. Bei den möglichen Formaten soll in dieser Arbeit in

- reellwertige Vektoren,
- strukturierte Textformate, sowie
- unstrukturierte Textformate

unterschieden werden.

Bei **reellwertigen Vektoren** sind als Datenobjekte beliebige Elemente x aus einem Vektorraum $\mathbb{R}^n$ gemeint. Dabei ist es nicht weiter relevant, in welcher Form diese konkret in übergeordneten Datenstrukturen eingebettet sind.

Ein Datum im **Textformat** hingegen, besteht aus beliebig vielen Wörtern w aus einem Alphabet Σ^*. Dieses Textdatum kann, je nach Art der systemabhängigen Zeichenrepräsentation (zum Beispiel *ASCII* oder *UTF-8*), in eine binäre Darstellung überführt werden. Die konkrete Umsetzung dieser Überführung soll aber im Weiteren nicht relevant sein. Bei Textformaten kann darüber hinaus in **strukturierte**, sowie **unstrukturierte** Texte unterschieden werden. Strukturierte Texte zeichnen sich dadurch aus, dass sie einer festgelegten maschinenlesbaren Grammatik (zum Beispiel in Form von standardisierten Schemata) unterliegen, in der jedes Wort einer bestimmten Bedeutung zugeordnet werden kann. Unter strukturierte Texte fallen daher zum Beispiel

xml-Dokumente, die bezüglich eines bestimmten Schemas entworfen wurden. Der Gegensatz dazu wird als *unstrukturiert* bezeichnet. Hierbei kann es sich um beliebige Texte, sprich eine willkürliche Folge von Wörtern w handeln.

2.2 Netzgraphen

Für verteilte Strukturen, in denen Datenobjekte von mehreren Entitäten verwaltet werden können, bieten sich als Abstraktion Netzgraphen an. Diese beschreiben ob und wie Entitäten untereinander in Verbindung stehen und welche Einschränkungen mit diesen Verbindungen verknüpft sind. In diesem Kapitel sollen daher Netzgraphen und deren Eigenschaften beschrieben werden.

2.2.1 Allgemeine Definitionen

Definition 2.2.1 (Allgmeiner Netzgraph)**.** Ein allgemeiner Netzgraph wird durch einen *ungerichteten* Graphen $G = (V, E)$ beschrieben, welcher nach Jungnickel [Jun07, S. 2] aus einer Knotenmenge $V = \{v_1, \dots v_n\} \neq \emptyset$ und einer Kantenmenge E besteht. E ist dabei eine Teilmenge aller 2-elementigen Teilmengen von V, d.h. $E \subseteq [V]^2$. Des weiteren wird jeder Kante $e_{ij} \in E(G)$ ein Kantengewicht $w : E \to \mathbb{R}$ zugewiesen. Ein Paar (G, w) bezeichnet dabei ein *Netzwerk* (s. Abbildung 2.2.1) mit $w(e)$ als Länge der Kante $e \in E(G)$ [Jun07, S. 59.]. $w(e)$ muss jedoch nicht zwangsweise einer Länge zwischen zwei Knoten entsprechen. Je nach Kontext der Darstellung kann es sich auch beispielsweise um Ähnlichkeiten handeln.

Zur Spezifikation von Graphen sind Adjazenz*listen* und Adjazenz*matrizen* geeignet. Bei der Verwendung von Adjazenzlisten werden für jeden Knoten eines Graphen, alle mit diesem adjazenten Knoten durch Listen erfasst. Sollen Graphen mittels Adjazenzmatrizen spezifiziert werden, so werden die Informationen über adjazente Knoten in einer Matrix organisiert. Diese Strukturen werden entsprechend nach Jungnickel [Jun07, S. 38] im Folgenden definiert.

Definition 2.2.2 (Adjazenzlisten)**.** Ein Graph $G = (V, E)$ mit einer Knotenmenge V(G)={1,. . . ,n}, wird beschrieben durch

1. die Anzahl an Knoten n, sowie
2. n Listen $A_1, \dots, A_n$, wobei A_i alle Knoten j enthält, sofern e_{ij} in $E(G)$ enthalten ist.

Definition 2.2.3 (Adjazenzmatrizen)**.** Die Knotenbeziehungen eines Graphen $G = (V, E)$ mit V(G)={1,. . . ,n}, werden in einer $(n \times n)$-Matrix A wie folgt dargestellt:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \dots & \dots & a_{nn} \end{pmatrix}, \qquad \text{(Adjazenzmatrix)}$$

wobei

$$a_{ij} = \begin{cases} 1 & \text{gdw. } (i,j) \in E(G), \\ 0 & \text{sonst.} \end{cases}$$

Über Adjazenzmatrizen lässt sich zudem sehr leicht der *Grad* (Anzahl ein-, bzw ausgehende Kanten) eines Knotens bestimmen, indem über entsprechende Zeilen bzw. Spalten von A aufsummiert wird. Handelt es sich um einen ungerichteten Graphen, so ist A symmetrisch und es gilt für den Grad

$$\deg(i) := \sum_{j=1}^{n} a_{ij} = \sum_{j=1}^{n} a_{ji} \tag{2.2.1}$$

Zudem gelte für die Kantengewichte $w(e_{ij}) \in \mathbb{R}$ eines Netzgraphen (G, w) nach Mohar [Moh97, S. 2]:

$$\begin{aligned} &w(e_{ij}) = w(e_{ji}) \ \forall i, j \in V, \\ &w(e_{ij}) \neq 0 \text{ gdw. } i \text{ und } j \text{ sind in } G \text{ adjazent.} \end{aligned} \tag{2.2.2}$$

$W = (w(e_{ij}))_{i,j=1,\dots,n}$ entspricht dabei der *gewichteten* Adjazenzmatrix bezüglich der Kantengewichte von G. Anstelle von $w(e_{ij})$ kann auch verkürzt w_{ij} geschrieben werden. Bei der Konstruktion von W (Gleichung (2.2.2)) wird ersichtlich, dass die Informationen einer gewichteten Adjazenzmatrix ausreichend sind, um daraus eindeutig eine allgemeine Adjazenzmatrix A abzuleiten. Dies geschieht, indem alle Einträge aus W, die ungleich 0 sind, in A auf 1 gesetzt werden, alle übrigen bleiben 0. Anders ausgedrückt

$$\begin{aligned} w(e_{ij}) \neq 0 &\Rightarrow a_{ij} = 1, \\ w(e_{ij}) = 0 &\Rightarrow a_{ij} = 0. \end{aligned}$$

Aus dem Umstand, dass Netzgraphen im allgemeinen ungerichtet sind (Definition 2.2.1), folgt die Symmetrie von A und W. Daher gilt

$$a_{ij} = a_{ji} \; \forall i, j \in V(G),$$

bzw. für einen gewichteten Netzgraphen

$$w_{ij} = w_{ji} \; \forall i, j \in V(G).$$

Für gewichtete Netzgraphen lässt sich, analog zu Gleichung (2.2.1), ebenfalls ein Knotengrad bestimmen:

$$\deg(i) := \sum_{j=1}^{n} w_{ij} \text{ [Lux07, S. 2]} \tag{2.2.3}$$

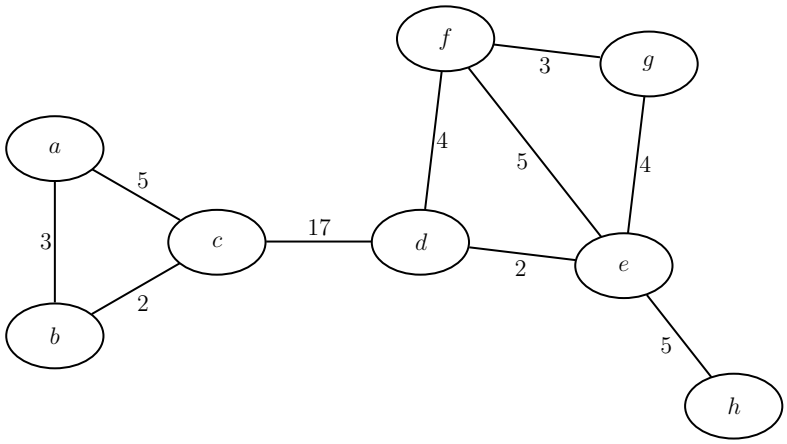


Abbildung 2.2.1: Beispiel für einen allgemeinen Netzgraphen G_{bsp} mit $|V(G_{bsp})| = 8$ und zufälligen Kantengewichten $w(e_{ij})$

Beispiel 2.2.1 (Netzgraph G_{bsp})**.** Der Netzgraph $G_{bsp} = (V_{bsp}, E_{bsp})$ (Abbildung 2.2.1) mit $|V_{bsp}| = 8$ wird durch das Aufstellen einer 8×8-Adjazenzmatrix, sowie einer entsprechenden gewichteten Adjazenzmatrix wie folgt beschrieben:

$$A_{bsp} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad \text{(Adjazenzmatrix)}$$

$$W_{bsp} = \begin{pmatrix} 0 & 3 & 5 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 5 & 2 & 0 & 17 & 0 & 0 & 0 & 0 \\ 0 & 0 & 17 & 0 & 2 & 4 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 3 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{pmatrix} \qquad \text{(gewichtete Adjazenzmatrix)}$$

2.2.2 Ungewichtete Netzgraphen

Ein Spezialfall der oben genannten Definition 2.2.1 stellt ein Netzgraph mit konstanten Kantengewichten $w(e_{ij}) = const$ dar. Somit liegt keine unterscheidbare Kantengewichtung vor, man spricht also von einem *ungewichteten Netzgraphen*. Für die Kantengewichte in einem solchen Graphen $G_u = (V, E)$ mit $i, j \in V(G_u)$ wird die Definition 2.2.1 wie folgt eingeschränkt:

Definition 2.2.4 (Kantengewichte in ungewichteten Netzgraphen)**.** Im Folgenden sei bei Netzgraphen ohne explizite Kantengewichte, eine einheitliche, konstante Gewichtung von

$$w(e) = 1 \ \forall e \in E(G_u)$$

festgelegt.

Somit gilt für das Beispiel 2.2.1

$$W_{bsp} = A_{bsp}.$$

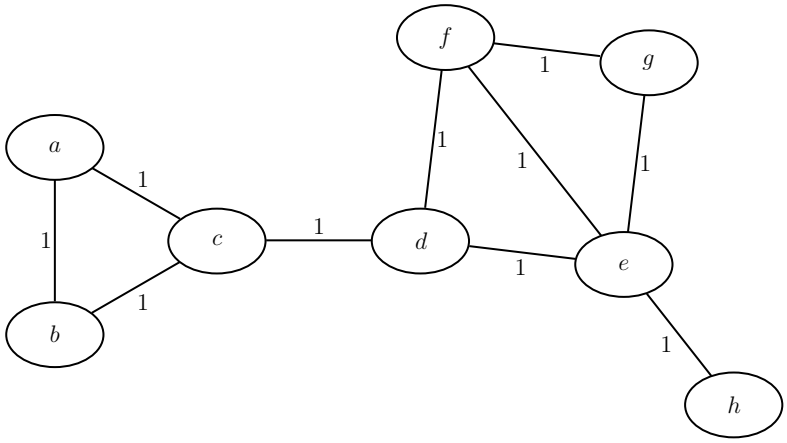


Abbildung 2.2.2: G_{u} mit konstanten Kantengewichten $w(e_{ij}) = 1$ für $(i, j) \in E_{\mathrm{u}}$

2.3 Abstände und Ähnlichkeiten

Bei der Verarbeitung von strukturierten Daten kann es nützlich sein, Vergleiche unter diesen anzustellen, oder diese in Kategorien einteilen zu können. Um dieses zu erreichen, gibt es zwei grundlegende Konzepte um Daten klassifizieren zu können. Zum einen **Distanzen**, die aussagen inwiefern zwei Datenobjekte *unähnlich* zueinander sind und zum anderen **Ähnlichkeiten** zwischen diesen. Im Folgenden sollen daher für Datenobjekte, oder auch allgemein gesprochen für Elemente aus einer beliebigen Datenmenge X bestimmte Festlegungen getroffen werden, die notwendig sind um eine Bestimmung von Distanzen, sowie Ähnlichkeiten zu ermöglichen.

2.3.1 Metriken

Metriken weisen Elementen eines Raumes einen nichtnegativen reellen Wert zu. Anschaulich kann es sich bei Metriken beispielsweise um Abstände zwischen zwei Elementen in einem Vektorraum über $\mathbb{R}$, oder auch die Pfadlängen in einem Graphen handeln.

Bei der Anwendung von Metriken auf Elemente, beziehungsweise Datenobjekte, lässt sich daher in

- reellwertige Vektoren $x_1, \ldots, x_n \in \mathbb{R}$,
- Knoten $v \in V(G)$ eines Netzgraphens,

oder auch beispielsweise in die

- syntaktische Unterscheidung von Formaten,
- sematische Kriterien, wie Kosten, Besitzer, Berechtigungen oder inhaltliche Ähnlichkeit, sowie
- Beziehungen zwischen chiffrierten Daten mit unbekanntem Inhalt

unterscheiden. Hierbei ist zu beachten, dass insbesondere bei den letztgenannten Punkten Abbildungen nötig sind, die diese Daten bezüglich ihres Typs in jeweils vergleichbare Kenngrößen überführen. Dabei ist in erster Linie eine Vergleichbarkeit von Daten mit identischem Typ gemeint, da ein Vergleich von unterschiedlichen Datentypen nur eingeschränkt sinnvoll ist. Eine Metrik muss dabei verschiedene Eigenschaften erfüllen, die im Weiteren dargelegt werden.

Definition 2.3.1 (Metrik)**.** Für eine Metrik $d : X \times X \to \mathbb{R}$ auf einer beliebigen Menge $X \neq \emptyset$ muss nach Forster [For11, S. 1] gelten, dass

$$d(x, y) = 0, \text{ gdw. } x = y \tag{2.3.1}$$

$$d(x, y) = d(y, x)\ \forall x, y \in X \qquad \text{(Symmetrie)} \tag{2.3.2}$$

$$d(x, z) \leq d(x, y) + d(y, z)\ \forall x, y, z \in X \qquad \text{(Dreiecksungleichung)} \tag{2.3.3}$$

daraus folgt zusätzlich

$$d(x, y) \geq 0\ \forall x, y \in X.$$

Handelt es sich um reellwertige Vektordaten, es liegen somit ausschließlich Daten im $\mathbb{R}^n$ vor, so lässt sich nach Forster [For11, S. 3f] eine Metrik für *normierte Vektorräume* V festlegen:

Definition 2.3.2 (Metrik über normierte Vektorräume)**.** Ist $(V, \| \ \|)$ ein normierter Vektorraum über dem Körper $\mathbb{K} = \mathbb{R}$, so definiert

$$d(x, y) := \|x - y\| \quad \text{mit } x, y \in V$$

eine Metrik auf V.

Aufgrund der Eigenschaften der *Norm* auf V gilt laut Forster [For11, S. 3f] die vorherige Definition 2.3.1 analog auch für normierte Vektorräume.

2.3.2 Ähnlichkeitsmaße

Im Gegensatz zu Distanzen, existieren auch Ähnlichkeitsmaße, die beschreiben inwiefern sich Daten, oder verallgemeinert Objekte, hinsichtlich bestimmter Kriterien ähneln.

Definition 2.3.3 (Ähnlichkeitsmaß)**.** Ein Ähnlichkeitsmaß muss nach Runkler [Run10, S. 12] folgende Bedingungen erfüllen:

$$s(x, y) = s(y, x) \tag{2.3.4}$$

$$s(x, y) \geq 0 \tag{2.3.5}$$

$$s(x, x) \geq s(x, y) \tag{2.3.6}$$

sowie für normalisierte Ähnlichkeitsmaße

$$s(x, x) = 1 \tag{2.3.7}$$

Zusammenhang von Abständen und Ähnlichkeiten Häufig sind Verfahren zur Verarbeitung von Datenobjekten auf deren Abstände, oder auch Ähnlichkeiten angewiesen. Daher kann es, je nach Art der vorliegenden Datenbeziehungen, gegebenenfalls notwendig sein, dass Ähnlichkeiten s in Abstände d und umgekehrt überführt werden müssen. Für **normalisierte** Ähnlichkeiten kann dies nach [Run10, S. 12] durch entsprechende Umformungen wie

$$d = 1 - s \text{ für } s \in [0, 1] \tag{2.3.8}$$

oder

$$d = 1/s - 1 \text{ für } s \in (0, 1] \tag{2.3.9}$$

erreicht werden. Durch Gleichung (2.3.8) werden normierte Ähnlichkeiten in ebenfalls normierte Distanzen zwischen 0 und 1 überführt. Gleichung (2.3.9) hingegen sorgt dafür, dass bei Gleichheit ($s = 1$) der Abstand zwar ebenfalls 0 ist, für $s \to 0$ jedoch die Distanz $d \to \infty$ geht.

2.3.3 Reellwertige Vektoren

Häufig werden durch Daten unterschiedliche Objekte mit bestimmten Merkmalen beschrieben, zum Beispiel in Form von Bauteilen (siehe Abbildung 2.3.1). In solchen Fällen bietet es sich an, die Merkmale eines Objektes in Form eines Vektors zusammen zu fassen, wobei die einzelnen Vektor-Komponenten als unterschiedliche Ausprägungen von Attributen eines bestimmten Datums interpretiert werden können. Liegen die Merkmalsinformationen, bzw. Komponenten zudem als reellwertige Zahlen x_i, y_i vor, so erhält man eine Menge an reellwertigen Vektoren, deren (Un-) Ähnlichkeiten durch Abstands- bzw. Ähnlichkeitsmaße bestimmt werden können. Ausgehend von der Definition 2.3.2

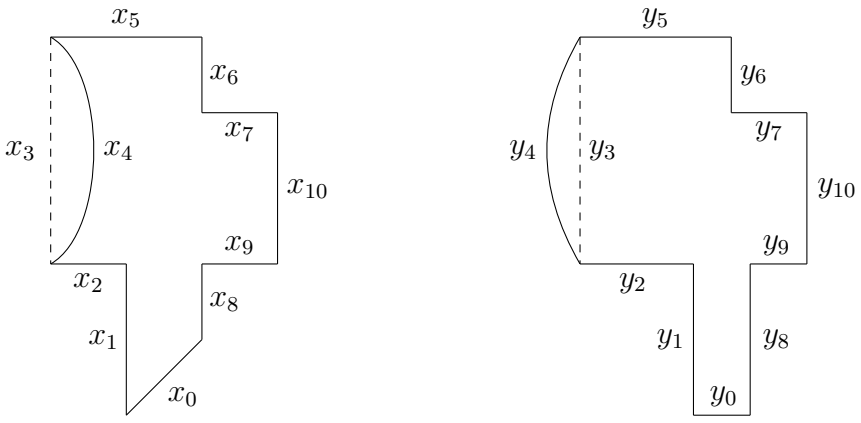


Abbildung 2.3.1: Beispiel für zwei Bauteile $x, y \in \mathbb{R}^{11}$ mit jeweils elf reellwertigen Merkmalen x_i und y_i

lassen sich folgende Abstandsmaße (2.3.10) bis (2.3.14) gemäß Ertel [Ert08,

S. 226f], sowie (2.3.15) nach Runkler [Run10, S. 14] für reellwertige Vektoren formulieren:

Definition 2.3.4 (Abstandsmaße reellwertiger Vektoren im $\mathbb{R}^n$)**.**

Euklidischer Abstand

$$d_e(x,y) = \|x-y\| = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}, \tag{2.3.10}$$

Summe der Abstandsquadrate

$$d_q(x,y) = d_e^2(x,y) = \sum_{i=1}^{n}(x_i-y_i)^2, \tag{2.3.11}$$

Manhattan-Abstand

$$d_m(x,y) = \sum_{i=1}^{n}|x_i-y_i|, \tag{2.3.12}$$

Maximumnorm

$$d_\infty(x,y) = \max_{i=1,\ldots,n}|x_i-y_i|, \tag{2.3.13}$$

normierter Skalarprodukt-Abstand

$$d_s(x,y) = \frac{\|x\|\|y\|}{xy}, \tag{2.3.14}$$

Hamming-Distanz

$$d_H(x,y) = \sum_{i=1}^{n} z(x_i,y_i) \tag{2.3.15}$$

$$\text{mit } z(x_i,y_i) = \begin{cases} 0 & \text{falls } x_i = y_i, \\ 1 & \text{sonst,} \end{cases}$$

für $x, y \in \mathbb{R}^n$.

Laut Ertel [Ert08, S. 195] kann es oftmals sinnvoll sein, einzelne Merkmale von Punkten aus $\mathbb{R}^n$ unterschiedlich stark zu gewichten. Um dieses zu errei-

chen wird von ihm ein *euklidisches Abstandsmaß mit Merkmalsgewichten* w_i eingeführt:

$$d_w(x,y) = \|x-y\| = \sqrt{\sum_{i=1}^{n} w_i(x_i-y_i)^2} \quad \text{mit } x,y,w \in \mathbb{R}^n. \tag{2.3.16}$$

Ähnlichkeiten zwischen reellwertigen Vektoren Die Abstandsmaße (2.3.10) bis (2.3.16) lassen sich durch Umformungen mittels Kehrwert, oder negiertem Abstand, in Ähnlichkeiten überführen. Ein weiteres Ähnlichkeitsmaß kann [nach Lux07, S. 408] durch die Verwendung einer Exponentialfunktion mit negiertem Abstand gebildet werden. Hierbei sorgt der Parameter σ dafür, dass benachbarte Punkte eine höhere Gewichtung erfahren können als weiter entfernte [Lux07, S. 408].

$$s_k(x,y) = \frac{1}{d(x,y)} \tag{2.3.17}$$

$$s_e(x,y) = -d(x,y) \tag{2.3.18}$$

$$s_g(x,y) = \exp\left(-\frac{d(x,y)}{(2\sigma)^2}\right) \tag{2.3.19}$$

Neben reellwertigen Vektoren besteht auch die Möglichkeit, Abstände zwischen den Knoten eines Netzgraphens zu messen. Daher lassen sich, analog zu oben genannten Gleichungen auch Ähnlichkeiten in Graphen bestimmen. Diese erhält man ausgehend von geeigneten Abstandsfunktionen, die in dem folgenden Abschnitt erläutert werden.

2.3.4 Netzgraphen

Gewichtungen in direkten Nachbarschaften Um die Abstände bzw. Ähnlichkeiten zwischen zwei beliebigen Knoten $i,j \in V(G)$ zu bestimmen, gibt es verschiedene Möglichkeiten. Ein einfacher Ansatz wäre es, nur die Kantengewichte von direkten Nachbarschaften, d.h. von adjazenten Knoten, zu betrachten.

Bei *nicht* verbundenen Kanten werden die Abstände $d^0_G(i,j)$ dann mit *unendlich*, sowie die Ähnlichkeiten $s^0_G(i,j)$ mit *null* bewertet.

Abstände zwischen $i, j \in V(G)$:

$$d_G^0(i,j) = \begin{cases} w(e_{ij}) & \text{für } e_{ij} \in E(G), \\ \infty & \text{sonst.} \end{cases} \tag{2.3.20}$$

Ähnlichkeiten zwischen $i, j \in V(G)$:

$$s_G^0(i,j) = \begin{cases} s_{ij} & \text{für } e_{ij} \in E(G), \\ 0 & \text{sonst.} \end{cases} \tag{2.3.21}$$

Hierbei stellt s_{ij} die Ähnlichkeit zwischen i und j dar. Diese kann entsprechend nach Abschnitt 2.3.2 aus d_{ij} errechnet werden.

Spezialfall: Ungewichtete Netzgraphen Der einfachste, aber sehr spezielle Fall, ist ein Netzgraph (G_u, w) der konstante Kantengewichte oder auch keine Informationen über diese enthält. Wie in Definition 2.2.4 festgelegt, gilt in einem solchen Fall $w(e) = 1\ \forall e \in E(G_u)$. Dies bedeutet, dass alle adjazenten Knoten aus (G_u, w) identische Abstände $d_{G_u}^0$, bzw. Ähnlichkeiten $s_{G_u}^0$ besitzen.

Pfadlängen in Netzgraphen Komplexer gestaltet sich hingegen die Bestimmung von $d(i,j)$ und $s(i,j)$, wenn auch *nicht* direkte Nachbarschaften berücksichtigt werden sollen. Hierbei können Abstände $d_G^P(i,j)$ bzw. Ähnlichkeiten $s_G^P(i,j)$ über *kürzeste Pfade* von i nach j berechnet werden. Nach Jungnickel [Jun07, S. 59] lassen sich Wege zwischen Knoten i und j als ein Pfad $\mathcal{W}_{ij} = (i, \ldots, j)$ darstellen, sofern sich i und j innerhalb einer Zusammenhangskomponente von (G, w) befinden. Die Länge von $\mathcal{W}_{ij}$ ist demnach

$$w(\mathcal{W}_{ij}) := w(e_{ik}) + \cdots + w(e_{lj}).$$

Daraus lässt sich als Distanz zwischen zwei Knoten i und j aus (G, w) folgern, dass

$$d_G^P(i,j) = \begin{cases} \infty & \text{wenn } j \text{ nicht von } i \text{ erreichbar,} \\ \min\{w(\mathcal{W}_{ij}) : \text{Pfad } \mathcal{W}_{ij} = (i, \ldots, j)\} & \text{sonst} \end{cases} \tag{2.3.22}$$

ist [nach Jun07, S. 60].

(Un-) Ähnlichkeiten von Knoten mittels kürzester Pfadlängen Sollen Unähnlichkeiten, also Distanzen $d_G^P(i,j)$, zwischen beliebigen Knoten abgebildet werden, so genügt die Anwendung von (2.3.22). Für die Bestimmung von Ähn-

lichkeiten $s_G^P(i,j)$ kann anschließend, wie in (2.3.17) bis (2.3.19) beschrieben, durch entsprechende Umformungen vorgegangen werden.

Liegt ein Netzwerk (G_u, w) ohne Kantengewichte vor (siehe Abschnitt 2.2.2), so lassen sich die kürzesten Wege zwischen zwei Knoten i und j durch eine *Breitensuche* [Jun07, S. 63f] mit der Laufzeit $\mathcal{O}(|E|)$ (siehe Algorithmus A.1) effizient berechnen. Hierbei entspricht der Abstand

$$d_{G_u}^P(i,j) = \sum_{w \in \mathcal{W}_{ij}} w = |\mathcal{W}_{ij}|,$$

da in (G_u, w) $w(e) = 1 \ \forall e \in E(G_u)$ gilt. Um nun eine Distanzmatrix W mit entsprechenden Kantengewichen $w(e_{ij})$ aufzustellen, muss für alle Kanten $e_{ij} \in V(G_u)$ eine Breitensuche durchgeführt werden. Hierdurch erhält man eine Komplexität von $\mathcal{O}(|V||E|)$ [Jun07, S. 84].

Bei Netzwerken mit beliebigen, nichtnegativen Gewichten $w \in \mathbb{R}^+$ bietet sich nach [Jun07, S. 76ff] der Algorithmus von *Dijkstra* an, der den kürzesten Pfad von Knoten i zu allen $j \neq i$ mit einer Laufzeit von $\mathcal{O}(|V|^2)$, modifiziert in $\mathcal{O}(|E| \log |V|)$ berechnet.

Der Algorithmus von *Floyd-Warshall* hingegen kann gemäß [Jun07, S. 84f] kürzeste Pfade auch in Graphen mit negativen Kantengewichten, d.h. bei $w \in \mathbb{R}$, in $\mathcal{O}(|V|^2)$ finden, sofern der Graph keine Zyklen negativer Länge aufweist.

Dies bedeutet, dass vor der Anwendung des Algorithmus von Dijkstra sichergestellt werden muss, dass keine Kanten mit negativen Längen, bzw. bei dem Einsatz von Floyd-Warshall keine negativen Zyklen vorhanden sind. Bezüglich der formalen Abläufe sei auf Algorithmus A.2 und Algorithmus A.3 (Anhang) verwiesen.

Infrastruktureigenschaften Da es sich bei Netzgraphen auch um eine abstrahierte Form eines Netzwerkes mit unterschiedlichen Teilnehmern handeln kann, bietet es sich an, neben einfachen Pfadlängen, bewertet mit (unter Umständen frei gewählten) Kantengewichten, auch konkrete bis hinzu physikalischen Eigenschaften der vorliegenden Infrastruktur zu betrachten. Hierbei können Faktoren wie

- Latenzen,
- Bandbreiten,
- Ausfallwahrscheinlichkeiten,
- monetäre Nutzungskosten oder auch

- Nutzungsberechtigungen

für bestimmte Pfade in Netzwerken relevant sein.

2.3.5 Inhaltliche Ähnlichkeit von strukturierten Datenobjekten

Um Daten bezüglich ihrer Inhalte gruppieren zu können, sind Verfahren notwendig, paarweise Abstände bzw. Ähnlichkeiten zwischen Datenobjekten zu berechnen. Dabei soll im Folgenden ein Datum d einen Text darstellen, welcher eine beliebige Anzahl Wörter w_i enthält. Nach Shawe-Taylor und Cristianini [STC04] wäre ein denkbarer Ansatz, jedes Datum durch einen reellwertigen Vektor mit festgelegter Länge darstellen zu können. Ein solcher Ansatz wird laut den Autoren als *bag-of-words*-Repräsentation bezeichnet. Hierbei wird für jeden Text d ein Vektor berechnet, welcher in jeder Dimension die Häufigkeit des Auftretens eines bestimmten Wortes angibt. Die Abbildung eines Textes auf einen solchen Vektor geschieht nach [STC04, S. 329] durch eine Abbildung

$$\phi : d \to \phi(d) = (\mathrm{tf}(w_1, d), \mathrm{tf}(w_2, d), \ldots, \mathrm{tf}(w_N, d)) \in \mathbb{R}^N,$$
$$\text{mit } \mathrm{tf}(w, d) = |\{w \mid w \in d\}|.$$

Mittels ϕ kann somit nach [STC04] ein *dictionary* erzeugt werden, welches alle zu vergleichenden Texte d auf entsprechende *bag-of-words*-Vektoren abbildet. Diese Vektoren lassen sich anschließend durch Verfahren, wie in Abschnitt 2.3.3 beschrieben, vergleichen. Der Nachteil eines solchen Vorgehens liegt allerdings zum einen in der hohen Dimensionalität der Vektoren, zum anderen kann die Reihenfolge der Wörter nicht berücksichtigt werden. Ist jedoch die Reihenfolge bzw. Position der Wörter ein entscheidendes Kriterium, so bietet sich die Verarbeitung mittels *String kernel*-Verfahren an, worauf im Folgenden eingegangen wird.

String kernel Die Idee des *String kernel*-Verfahrens beruht darauf, alle n-Gramme (Substrings der Länge n) eines Textes d_i und d_j zu bestimmen und diese zu vergleichen. Stimmen in beiden Texten viele Substrings überein, so wird davon ausgegangen, dass d_i und d_j ähnlich zueinander sind. Die Kernelfunktion K ist nach [Lod+02, S. 421] definiert als ein Skalarprodukt

$$K(d_i, d_j) = \langle \phi(d_i), \phi(d_j) \rangle,$$
$$\text{mit } \phi : D \to F.$$

ϕ beschreibt dabei eine Abbildung, die die zu vergleichenden Texte d_i und d_j in einen Merkmalsvektor

$$\phi(d) = (\phi_1(d), \ldots, \phi_N(d)) \in \mathbb{R}^N$$

überführt [Lod+02, S. 422]. Konkret bedeutet dies, dass für jeden Substring $u \in \Sigma^n$ eines Textes d der Länge n, die u Koordinate $\phi_u(d)$ definiert ist als

$$\phi_u(d) = \sum_{i:u=d[i]} \lambda^{l(i)} \text{ [Lod+02, S. 423]},$$

mit der Subsequenz u bezüglich der Indizes aus i und der Länge $l(i)$ der Subsequenz. $\lambda \in (0, 1)$ bezeichnet hierbei nach [Lod+02] einen *decay*-Faktor, der entsprechend der Länge der in dem Text vorkommenden Substrings potenziert wird. Zur Veranschaulichung wird dieses Verfahren in Beispiel 2.3.1, entnommen aus [Lod+02, S. 423], dargestellt.

Beispiel 2.3.1 (*String kernel*-Verfahren)**.** In diesem Beispiel soll das Verfahren anhand der Texte *cat, car, bat* und *bar* für eine Substringlänge von $n = 2$ illustriert werden. Dies wird durch die folgende Tabelle 2.3.1 dargestellt:

	c-a	c-t	a-t	b-a	b-t	c-r	a-r	b-r
ϕ(cat)	λ^2	λ^3	λ^2	0	0	0	0	0
ϕ(car)	λ^2	0	0	0	0	λ^3	λ^2	0
ϕ(bat)	0	0	λ^2	λ^2	λ^3	0	0	0
ϕ(bar)	0	0	0	λ^2	0	0	λ^2	λ^3

Tabelle 2.3.1: Beispiel für das String kernel-Verfahren anhand der Worte *cat*, *car*, *bat* und *bar* [Lod+02, S. 423]

Somit ergeben sich beispielsweise für *car*, *cat* und *bat* mit $\lambda = 0{,}1$ normalisierte Ähnlichkeiten von

$$K(\text{car}, \text{cat}) = \frac{\lambda^4}{2\lambda^4 + \lambda^6} = \frac{1}{2 + \lambda^2} \approx 0{,}4975$$
$$K(\text{car}, \text{bat}) = \frac{0}{2\lambda^4 + \lambda^6} = 0$$

Normalized Compression Distance Erschwerend bei der Analyse von textuellen Daten ist der Umstand, dass häufig Daten in unterschiedlichen Formaten zu verschiedenen Themenbereichen vorliegen und nicht so einfach in eine

Vektordarstellung überführt werden können. In einem solchen Fall kann das Verfahren der Normalized Compression Distance (NCD), wie es von Li u. a. [Li+01] vorgestellt wurde, eine Möglichkeit darstellen, Informationen über die Distanzen, sprich Unähnlichkeiten, zwischen zwei Datenobjekten zu gewinnen. Die Idee hierbei ist, dass die Kompression von ähnlichen Texten in ähnlich langen Ausgabe-Strings resultiert. Werden nun diese Kompressionslängen, wie in Gleichung (2.3.23) [Li+01, S. 3258] dargestellt, miteinander Verrechnet, so erhält man eine Aussage über die relative Distanz zwischen zwei Datenobjekten.

$$NCD(x,y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}} \tag{2.3.23}$$

Die Autoren der NCD weisen darauf hin, dass zur Berechnung von Ähnlichkeiten ein eng verwandter Ansatz bei [KLR04, S. 3] zu finden ist. Hierbei kann durch das sogenannte Compression-based Dissimilarity Measure (CDM) (Gleichung (2.3.24)) eine Aussage darüber getroffen werden, wie sehr sich zwei Ressourcen ähneln.

$$CDM(x,y) = \frac{C(xy)}{C(x) + C(y)} \tag{2.3.24}$$

Zur Kompression $C(x)$, $C(y)$, sowie der Konkatenation von x und y, $C(xy)$, bieten sich laut den Autoren Verfahren wie *bzip2* oder auch *gzip* an.

2.3.6 Meta-Informationen

Neben dem eigentlichen Inhalt können auch weitere Informationen einem Datenobjekt zugeordnet sein. So können für ein Datenobjekt unter anderem die jeweiligen

- Kosten,
- Besitzer,
- Nutzungsberechtigungen,
- Erstellungs-, Zugriffs-, Änderungszeitpunkte sowie
- Objektgrößen

zusätzlich hinterlegt werden, welche in unterschiedlichem Grad vom Inhalt des zugehörigen Datums abhängen. Aus diesem Grund kann es sinnvoll sein, auch diese als *Meta-Informationen* bezeichneten Attribute, bei einer Ähnlichkeitsanalyse zu berücksichtigen.

2.4 Hash-Funktionen

Eine Hash-Funktion h ist eine Abbildung die von [MVOV97, S. 33] definiert wird als

> [...] computational efficient function mapping binary strings of arbitrary length to strings of some fixed length [...]

Diese bildet also eine Urbildmenge X injektiv auf eine Zielmenge Y der Form $h : X \to Y$ ab. Hierbei gelten in Anlehnung an [Mer90, S. 430] und [Pre98, S. 62] allgemein folgende Bedingungen:

1. h kann auf beliebig lange Argumente $x \in X$ angewendet werden. (2.4.1)

2. Alle von h erzeugten Elemente $y \in Y$ haben eine feste Länge n. (2.4.2)

3. Für gegebenes h und $x \in X$ ist die Berechnung von $h(x)$ leicht. (2.4.3)

4. Für gegebenes h und $y \in Y$ ein x zu finden, so dass $y = h(x)$ gilt, ist *schwer berechenbar.* (2.4.4)

5. Für gegebenes h ein Paar x, x' zu finden, so dass $x \neq x'$ und $h(x) = h(x')$ gilt, ist *schwer berechenbar.* (2.4.5)

Die Bezeichnung *schwer berechenbar* kann allerdings ein, abhängig vom Betrachter, dehnbarer Begriff sein. Aus diesem Grund bezeichnet [Mer90, S. 431] Hash-Funktionen als schwer berechenbar, wenn diese eine hinreichend große Ausgabelänge aufweisen und somit ein nicht vertretbar hoher Aufwand bei der Suche nach einem x' für ein gegebenes x entsteht.

Hash-Funktionen, die die Bedingung (2.4.4) bzw. (2.4.5) erfüllen, werden zudem nach [ZMI91, S. 285f] als *„universal one-way hash functions“*, bzw. *„collision intractable hash functions“* bezeichnet.

2.4.1 Gleichverteiltes Hashing

Bei Hash-Funktionen, wie zum Beispiel dem *Secure Hash Algorithm 1 (SHA-1)*, die bei kryptographischen Signaturen, oder der Identifikation von Ressourcen in Peer-to-Peer-Systemen eingesetzt werden, erfolgt die Abbildung $h : X \to Y$ von einem Datum $x \in X$ auf einen Hash-Wert $y \in \{y_1, \ldots, y_{2^n}\} = Y$ gleichverteilt. Geht man von einer festen binären Ausgabelänge n von h aus, so beträgt

demnach die Wahrscheinlichkeit, dass ein beliebiges x einem bestimmten y zugeordnet wird, genau $p(x) = 1/2^n$ für alle $x \in X$.

Dieser Umstand begründet sich durch die Anforderungen an Hash-Funktionen, die in dem oben genannten Kontext eingesetzt werden:

1. **Kollisionssicherheit:**
   Soll eine Identifiziertbarkeit von Ressourcen anhand eines Hash-Wertes gewährleistet werden, so muss sichergestellt werden, dass trotz der festen Länge des Hash-Wertes, die Zuordnung von Ressource auf Identifikator möglichst eindeutig ist, um widersprüchliche Zuordnungen wie $h(x') = y = h(x)$ zu vermeiden. Aus diesem Grund werden Eingaben x somit auf ein $y_k \in Y$ abgebildet, so dass alle y_k mit gleicher Wahrscheinlichkeit in Frage kommen.

2. **Unumkehrbarkeit:**
   Gerade bei kryptographischen Signaturen, soll die Echtheit bzw. Manipulationsfreiheit eines Dokumentes bestätigt werden. Zu diesem Zweck werden Prüfsummen mittels Hash-Funktionen gebildet. Die Idee dabei ist, dass bereits kleine Änderungen am Dokument signifikante Änderungen bei der Prüfsumme hervorrufen. Um dieses zu gewährleisten, darf es (mit vertretbarem Aufwand) **nicht** möglich sein, zu gegebenen $h(x)$ ein $h^{-1}(x) = x$ zu errechnen.

2.4.2 Locality Sensitive Hashing

Locality-Sensitive Hashing (LSH) beschreibt in puncto Kollisionssicherheit das genaue Gegenteil von gleichverteilten Hash-Funktionen (Abschnitt 2.4.1). Ziel dabei ist es, dass ähnliche Werte x auf den gleichen Hash-Wert y abbilden.

Um dieses zu erreichen werden Hash-Funktionen $g_i \in \mathcal{H}$ auf die Datenmenge $X \subset \mathbb{R}^n$ angewendet, die ähnliche Daten auf jeweils gleiche Hash-Werte h_i abbilden. Durch den Einsatz mehrerer (l) Hash-Funktionen soll eine Identifizierung ähnlicher Datenpunkte x erleichtert werden. Hierzu werden die zu hashenden Vektoren zuerst in eine *unäre* Darstellung überführt. So wird beispielsweise aus dem Vektor $x = (1, 5, 4)$ die unäre Repräsentation

$$u_x = (\underbrace{10000}_{x_1}\ \underbrace{11111}_{x_2}\ \underbrace{11110}_{x_3}) \tag{2.4.6}$$

durch eine Verkettung von x_i Einsen gefolgt von

$$C = \max(x) - x_i$$

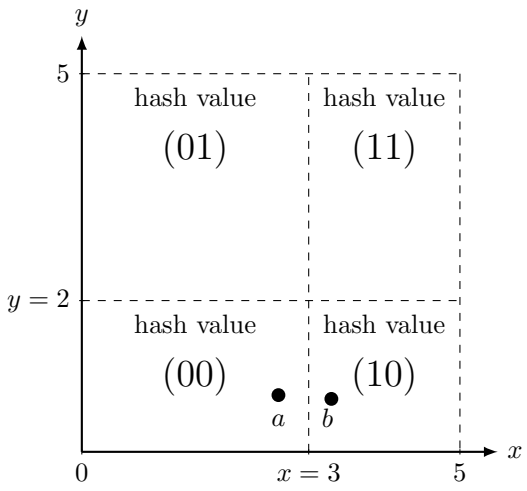


Abbildung 2.4.1: Partitionierung eines Vektorraums durch eine LSH-Funktion [KIW04, S. 117]

Nullen. Dieser Schritt ist notwendig für die Bestimmung der Hash-Funktionen g_i, welche Vektoren in unärer Darstellung auf einen binären Vektor mit fester Länge k abbilden:

$$g : u_x \to \{0,1\}^k \tag{2.4.7}$$

Hierbei werden für jedes g_i zufällig k Indizes aus den unären Repräsentationen der Vektoren x gewählt und deren Werte konkateniert.

$$g(u) = (u_{J(1)} \circ u_{J(2)} \circ \cdots \circ u_{J(k)})$$
$$\text{mit } J = \{k \text{ zufällig gewählte Indizes aus } u\}$$

Durch die feste Ausgabelänge k wird nach [KIW04, S. 117] *ein Vektorraum in Zellen durch k Hyperebenen aufgeteilt*, welcher in der Abbildung 2.4.1 für $k = 2$ und einer maximalen Vektordimension von $C = 5$ dargestellt ist. Hierbei wurde von den Autoren eine Hash-Funktion g gewählt, die das dritte Bit ($x = 3$) sowie das siebte Bit ($y = 2$) eines unären Vektors zur Bestimmung des Hash-Wertes verwendet. Jeder Vektor besteht in diesem Beispiel aus insgesamt 10 Bits, jeweils 5 für jede Dimension.

Für das eingangs gewählte Beispiel von $x = (1, 5, 4)$ ist der Hash-Vorgang wie in Abbildung 2.4.2 mit einer zufällig gewählten Hash-Funktion dargestellt. Hierbei wurden zur Bestimmung von $g(u_x)$ die Bits $3, 7, 9$ und 12 gewählt.

Die Ähnlichkeit zweier Vektoren kommt dadurch zum Ausdruck, dass die von den Hash-Funktionen gewählten Bits in der unären Repräsentation der Vektoren gleich sind.

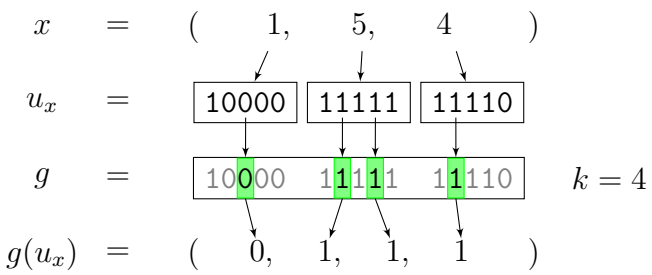


Abbildung 2.4.2: Exemplarisches LS-Hashing

2.5 Reellwertige Daten als Graphen

In vielen Fällen kann es sinnvoll sein, Datenpunkte in eine Graphenstruktur zu überführen. Gründe können zum Beispiel darin liegen, dass Nachbarschaften, oder allgemein Beziehungen zwischen einzelnen Datenpunkten betrachtet werden sollen. Um eine solche Überführung von Daten aus dem $\mathbb{R}^n$ in einen Netzgraphen G zu ermöglichen, gibt es verschiedene Ansätze, die im Folgenden vorgestellt werden. Dabei haben alle Verfahren gemein, dass jeder Datenpunkt $x_i \in \mathbb{R}^n$ im späteren Netzgraphen einem Knoten $i \in V(G)$ entspricht, entsprechend der Abbildung $\mathbb{R}^n \to V(G)$. Zudem werden die verschiedenen Ansätze durch Beispiele illustriert, die sich zur besseren Vergleichbarkeit jeweils auf die gleiche Vektormenge $a, \ldots, j \in \mathbb{R}^2$ beziehen.

2.5.1 Vollständiger Graph

Ein einfacher, aber auch naiver Ansatz ist, alle Datenpunkte in einen vollständigen Graphen K_n mit $V(K_n) = \{1, \ldots, n\}^2$ zu transformieren. Dabei wird jeder Knoten i mit jedem übrigen Knoten j aus $V(G)$ verbunden, wodurch ein Graph mit insgesamt $n(n-1)$ Kanten entsteht. In der Abbildung 2.5.1 wird dieser Vorgang beispielhaft für zehn Vektoren dargestellt. Bei diesem Verfahren entstehen Graphen, die durch eine dicht besetzte Adjazenz-, bzw. Ähnlichkeits- oder Abstandsmatrix repräsentiert werden. Da bei Clustering-Verfahren die paarweisen Ähnlichkeiten von nah gelegenen Datenobjekten von Interesse sind, eignet sich nach Luxburg [Lux07, S. 396] bei vollständigen Graphen eine *Gauß*'sche Ähnlichkeitsfunktion

$$s(x, y) = \exp\left(\frac{-||x - y||^2}{2\sigma^2}\right),$$

bei der der Parameter σ einen vergleichbaren Einfluss wie ε in Abschnitt 2.5.2 auf die Größe lokaler Nachbarschaften hat. Durch die Wahl einer solchen Funktion gehen die Ähnlichkeiten von sehr weit entfernten Punkten, aufgrund der Eigenschaften der Exponentialfunktion, schnell gegen 0.

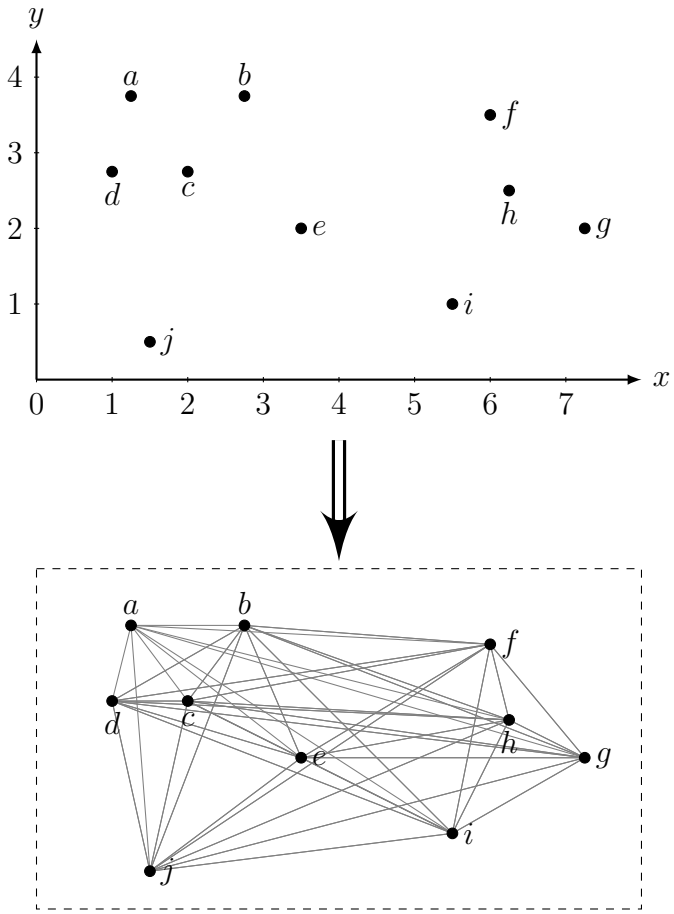


Abbildung 2.5.1: Beispiel für die Überführung der Datenpunkte $a, \ldots, j \in \mathbb{R}^2$ in einen vollständigen Graphen

2.5.2 ε-Neighborhood

Bei diesem Ansatz wird im Gegensatz zum *vollständigen Graphen* (Abschnitt 2.5.1) versucht auszuschließen, dass sehr weit entfernte Punkte durch Kanten verbunden werden, da diese vermutlich nicht zu einer Nachbarschaft gehören. Hierfür wird für alle Datenpunkte aus $\mathbb{R}^n$ ein *„Sichtradius“* $\varepsilon > 0$ definiert, welcher im Laufe des Verfahrens konstant bleibt. Anschließend wird nur dann ein Punkt x_i mit x_j verbunden, wenn diese sich „sehen“ können, also deren euklidischer Abstand nicht größer als ε ist. Dies wird in der Abbildung 2.5.2 entsprechend für Vektoren im $\mathbb{R}^2$ visualisiert, wobei zur besseren Übersicht nur die ε-Umgebungen um die Punkte c und h dargestellt sind.

2.5.3 k-Nearest Neighbor

Das Verfahren des k-Nearest Neighbor (kNN) bestimmt für jeden Datenpunkt x_i seine k nächstgelegenen Nachbarn und verbindet diese mit einer gerichteten Kante, ausgehend von x_i. Hierbei können also, je nach Anordnung der Vektoren, im Prinzip beliebig große Distanzen überbrückt werden. Das bedeu-

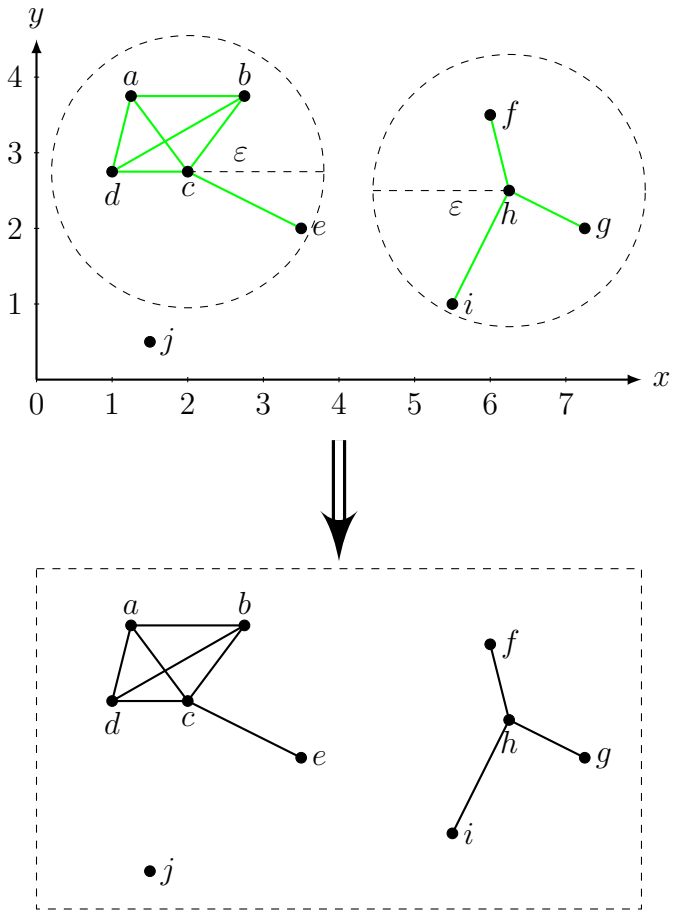


Abbildung 2.5.2: Beispiel für die Überführung der Datenpunkte $a, \ldots, j \in \mathbb{R}^2$ in einen ε-Neighborhood-Graphen mit $\varepsilon = 1{,}8$

tet, es kann Punkte auf beliebigen Skalen [Lux07, S. 408] verbinden. Bei dem ε-Nachbarschaftsgraphen hingegen (Abschnitt 2.5.2), stellte das ε eine obere Schranke für die Länge der möglichen Verbindungen von Vektoren dar.

Nach [Lux07, S. 408] führt dieses Verfahren dazu, dass sehr dichte Regionen an Vektordaten zu entsprechend dichten Subgraphen führen und falls diese Regionen genügend weit voneinander entfernt sind, der Graph in solche Subgraphen als Zusammenhangskomponenten zerfällt. Dieser Ansatz zur Überführung von Vektoren in Graphen wird in der Abbildung 2.5.3 veranschaulicht.

2.5.4 Mutual k-Nearest Neighbor

Mutual k-Nearest Neighbor (MkNN) unterscheidet sich dahingehend von kNN, dass genau dann Kanten zwischen zwei Knoten i, j gezogen werden, wenn x_i zu den k-nächsten Nachbarn von x_j **und** x_j zu den k-nächsten Nachbarn von x_i gehört. Dies hat nach Ansicht von [Lux07, S. 408] zur Folge, dass zwar Bereiche mit konstanter Dichte verbunden werden, jedoch Bereiche mit unterschiedlicher

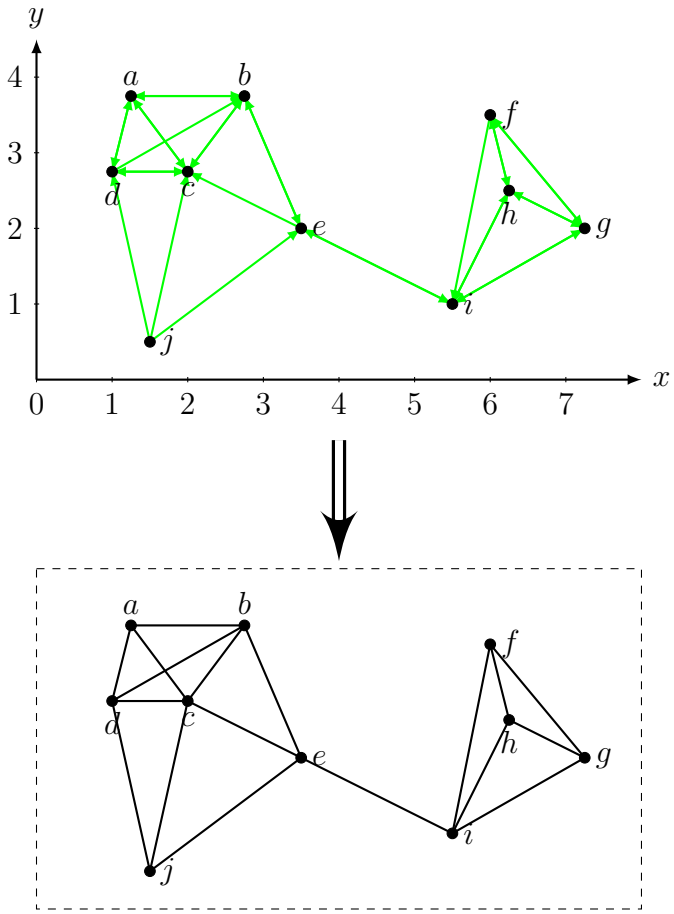


Abbildung 2.5.3: Beispiel für die Überführung der Datenpunkte $a, \ldots, j \in \mathbb{R}^2$ in einen kNN-Graphen mit $k = 3$

Dichte nicht. Somit sei laut der Autorin MkNN „zwischen“ ε-Nachbarschaften und kNN einzuordnen. Bei den Ergebnissen der MkNN, die in der Abbildung 2.5.4 dargestellt sind, fällt daher auf, dass im Gegensatz zur Abbildung 2.5.3 der Knoten j nun isoliert ist, da keiner der übrigen Knoten ihn zu seinen drei nächstgelegenen zählt.

2.5.5 Approximative k-Nearest Neighbor

Um einen kNN-Graphen wie in dem obigen Abschnitt 2.5.3 zu erstellen, ist im Allgemeinen ein Aufwand von $\mathcal{O}(n^2 \log n)$ nötig, da für jeden Datenpunkt p_i die Distanz zu allen Datenpunkte p_j mit $i \neq j$ bestimmt und unter diesen die k kleinsten ausgewählt werden müssen. Dies hat zur Folge, dass sich gerade bei sehr großen Datensätzen die quadratische Laufzeit signifikant bemerkbar macht. Das Ziel des im Folgenden erläuterten *Approximative* k*-Nearest Neighbor (AkNN)* Verfahrens liegt darin, kNN-Graphen zu erzeugen, die in unter quadratischer Laufzeit operieren und dabei möglichst geringe Qualitätseinbußen im Vergleich zu exakten kNN-Graphen aufweisen.

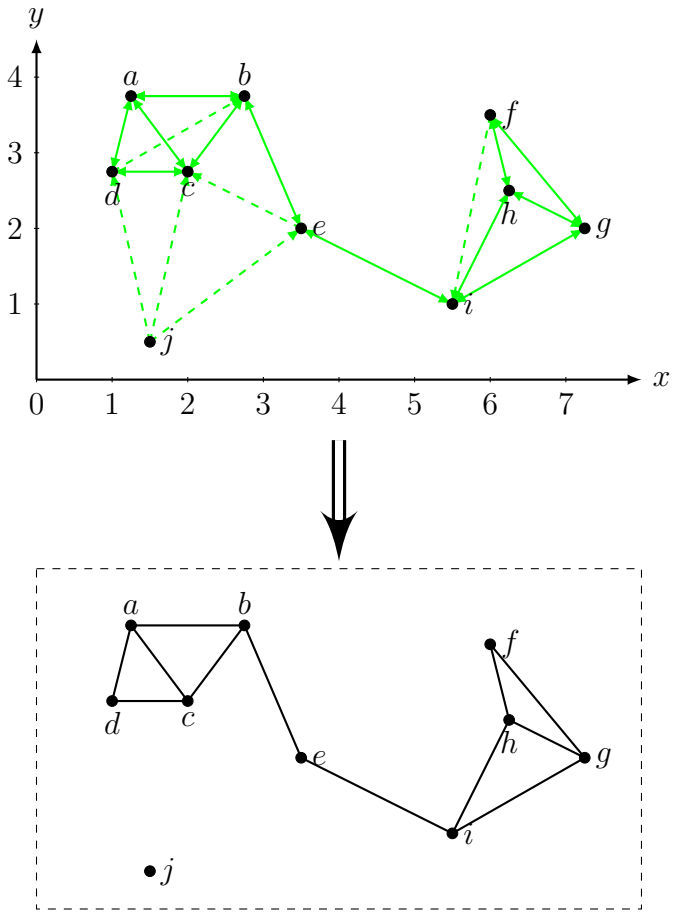


Abbildung 2.5.4: Beispiel für die Überführung der Datenpunkte $a, \ldots, j \in \mathbb{R}^2$ in einen MkNN-Graphen mit $k = 3$

Um einen solchen Graphen zu erzeugen, ist das in Abschnitt 2.4.2 vorgestellte LSH-Verfahren geeignet. Hierbei müssen im Voraus bestimmte Parameter festgelegt werden. Diese umfassen

- die Anzahl zu verwendenden Hash-Funktionen l mit $g_1, \ldots, g_l \in \mathcal{H}$, sowie
- deren gewünschte Ausgabelänge $\widetilde{k}$.

Dabei ist die Wahl des Parameters l von besonderer Bedeutung, da durch die Verwendung mehrerer Hash-Funktionen sichergestellt werden soll, dass benachbarte Vektoren auch als solche identifiziert werden. Betrachtet man hierzu das Beispiel in Abbildung 2.4.1, so fällt auf, dass bei der Verwendung von der dort gewählten LSH-Funktion die Punkte a und b in unterschiedliche Zellen fallen. Dies bedeutet im Kontext des LSH, dass diese nicht benachbart, sprich unähnlich sind. Um solche *false-negatives* zu vermeiden, werden mehre-

re Hash-Funktionen verwendet. Die Idee dabei ist, dass wenn für eine weitere Hash-Funktion g'

$$g'(a) = g'(b)$$

gilt, beide Punkte als ähnlich gelten. Somit genügt es, wenn mindestens eine Funktion $g_i \in \mathcal{H}$ diese korrekt als benachbart identifiziert. Allerdings weisen [KIW04, S. 117] darauf hin, dass bei einer zu großen Wahl von l die Gefahr besteht, dass weit entfernte Punkte fälschlicherweise als *false-positive* Nachbarn identifiziert werden könnten.

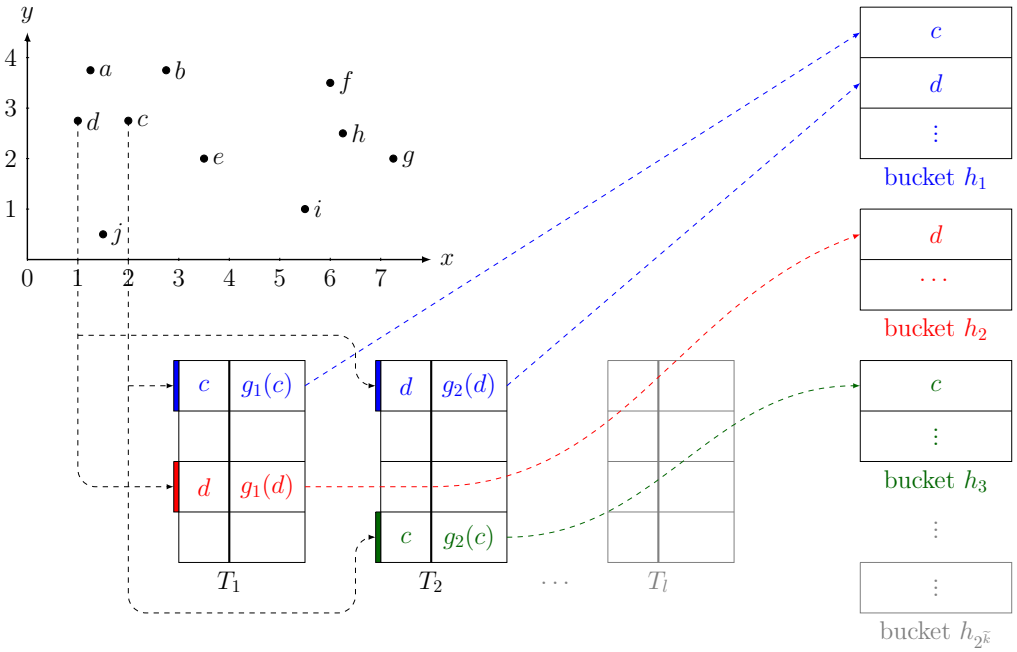


Abbildung 2.5.5: Locality-Sensitive Hashing am Beispiel von zwei Datenpunkten

Es wird, wie in Abbildung 2.5.5 exemplarisch dargestellt, für jeden Datenpunkt x sein jeweiliger Hash-Wert $h_i = g_i(p)$ berechnet. Aus den daraus resultierenden Hash-Tabellen

$$T_i = \{h \mid h = g_i(p)\},\ \forall i \in I$$

werden einzelne *Buckets* B_h wie folgt abgeleitet:

$$B_h = \{p \mid h = g_i(p)\},\ \forall h \in H$$
$$\text{und } g_i \in \mathcal{H} \text{ Menge aller LSH-Funktionen}$$
$$\text{mit } I = \{1, \ldots, l\},\ h = \{0,1\}^{\tilde{k}},\ p \in \mathbb{R}^n,\ g : \mathbb{R}^n \to H$$

Jeder Hash-Wert $h = g_i(p_j)$ bezeichnet **genau** einen *Bucket*. Ein Bucket (engl. Eimer) bezeichnet dabei die Menge an Datenpunkten die den gleichen Hash-Wert $h_i \in H$ aufweisen. Es werden daher alle Datenpunkte p_j in die Buckets eingefügt, die für den jeweiligen Hash-Wert h_i zuständig sind.

Benachbarte Punkte zu einem beliebigen *Query* q lassen sich finden, indem dieses, gemäß dem oben beschriebenen Verfahren, gehasht und den entsprechenden Buckets zugewiesen wird. Man erhält nun die zu q ähnlichen Datenpunkte, indem alle Bucket-Mengen die q enthalten vereinigt werden:

$$\mathrm{AkNN}_q = \bigcup_{h=g_i(q)} B_h$$

Somit ergibt sich nach [KIW04, S. 121] für n Datenpunkte, l Hash-Funktionen, sowie einer maximalen Bucket-Größe $|B|$ eine Laufzeit von $\mathcal{O}(nl|B|) < \mathcal{O}(n^2)$.

Der große Vorteil dieses Verfahrens liegt darin, dass bei der Bestimmung der k nächsten Nachbarn große Teile des Datenbestandes nicht betrachtet werden müssen, da nach Abschluss der initialen Hash-Phase, alle in Frage kommenden Kandidaten sich in den gleichen Buckets befinden wie das *Query*. Zudem lassen sich bei jeder individuellen Anfrage die Buckets erneut verwenden, so dass deren Berechnung nur einmal vorgenommen werden muss.

2.6 Vorverarbeitung ungewichteter Netzgraphen

Für Netzgraphen $G = (V, E)$ kann der Fall eintreten, dass keine Kantengewichte bekannt sind oder keine bekanntgegeben wurden. Sollen jedoch in solchen ungewichteten Graphen lokale Nachbarschaften (*Cluster*) identifiziert werden, so sind die alleinigen Informationen darüber, welche zwei Knoten v_i und v_j adjazent sind, nicht unbedingt ausreichend. Vielmehr ist es hilfreich, zusätzlich die Gewichtungen $w(e_{ij})$ zu kennen, um genauere Aussagen über den Beziehungsgrad zweier Knoten treffen zu können. Eine solche Gewichtung kann bei Graphen, basierend auf reellwertigen Vektoren im $\mathbb{R}^n$, bereits bei der Erzeugung leicht vorgenommen werden. Bei Graphen mit gleichgewichteten Knotenbeziehungen muss hierfür **vor** dem eigentlichen Clustering eine *Vorverarbeitung* durchgeführt werden, um eine möglichst gute Abschätzung der Beziehungen treffen zu können. Eine solche Berechnung von Kantengewichten kann mittels Common Neighborhood Subgraph Density (CND) erfolgen, die im folgenden Abschnitt beschrieben wird.

2.6.1 Common Neighborhood Subgraph Density

Das Verfahren der Common Neighborhood Subgraph Density (CND) [KC09] versucht Aussagen über Netzgraphen (G_u, w) ohne bekannte, oder nicht vorhandene Kantengewichte, bezüglich Gruppierungen von Knoten zu treffen. Das Ziel ist, Bereiche mit einer hohen Kantendichte in Gemeinschaften zusammenzufassen, so dass Bereiche ausserhalb dieser möglichst dünn vernetzt sind. Hierbei wird für alle Kanten $e_{ij} \in E(G_u)$ das Kantengewicht $w(e_{ij})$ über die von Kang und Choi [KC09, S. 177] beschriebene Kantendichte K_{ij} bestimmt. K_{ij} berechnet sich dabei über einen Sub-Graphen gemeinsamer Nachbarn von i und j wie folgt:

$$K_{ij} = \frac{\sum_{k \in \mathcal{C}_{ij}} \sum_{l \in \mathcal{C}_{ij}} A_{kl}}{|\mathcal{C}_{ij}|(|\mathcal{C}_{ij}| - 1)}, \tag{2.6.1}$$

mit der Indexmenge $\mathcal{C}_{ij}$ der gemeinsamen Nachbarn von i und j

$$\mathcal{C}_{ij} = \{i, j\} \cup \{k | A_{ik} \neq 0 \text{ und } A_{jk} \neq 0\}. \tag{2.6.2}$$

Hierbei müssen einzelne Knotenpaare i, j nicht zwangsläufig adjazent sein, damit für diese eine Kantendichte bestimmt werden kann. In einem solchen Fall ($e_{ij} \notin V(G)$) erhält man trotzdem einen Indikator, der aussagt, ob i und j zu einer Gemeinschaft gehören [KC09, S. 177].

Betrachtet man nun aus Abbildung 2.2.2 die beiden Subgraphen G' und G'' (Abbildung 2.6.1), die durch die Knotenteilmengen $V' = \{a, b, c\}$, sowie $V'' = \{c, d\}$ aufgespannt werden, so fällt auf, dass die Kantengewichte für beide Subgraphen jeweils *1* sind. Dies folgt aus der Tatsache, dass sowohl G', als auch

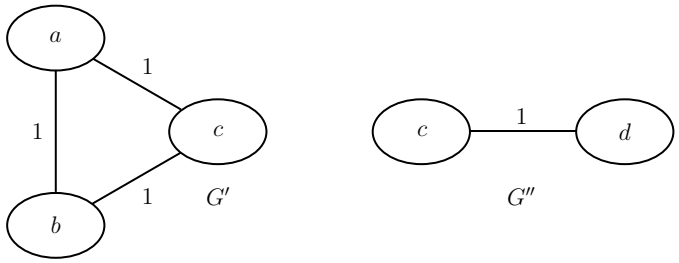


Abbildung 2.6.1: Subgraphen G' und G'' aus G_u mit $V' = \{a, b, c\}$ und $V'' = \{c, d\}$, sowie Kantengewichten K_{ij}.

G'' eine Clique darstellen und somit der Quotient aus der Anzahl vorhandener Kanten (ungerichtete Kanten werden hierbei doppelt gezählt) und maximal möglicher Kanten *1* ist. Um nun zu vermeiden, dass Verbindungen zwischen zwei Subgraphen genauso gewichtet werden wie Cliquen, wird von Kang und Choi [KC09] die Gleichung (2.6.1) modifiziert, indem K_{ij} zusätzlich mit der

potenzierten Anzahl der gemeinsamen Nachbarn von i und j multipliziert wird, wodurch sich

$$\bar{K}_{ij} = |\mathcal{C}_{ij}|^{\gamma} \frac{\sum_{k \in \mathcal{C}_{ij}} \sum_{l \in \mathcal{C}_{ij}} A_{kl}}{|\mathcal{C}_{ij}|(|\mathcal{C}_{ij}| - 1)}, \tag{2.6.3}$$

ergibt. Laut Kang und Choi [KC09, S. 179f] hat sich hierbei ein Exponent γ von 0,1 bis 3 als brauchbar erwiesen. Optimale Ergebnisse bei der Erkennung von Gemeinschaften wurden dabei mit $\gamma = 1$ erzielt [KC09, S. 181].

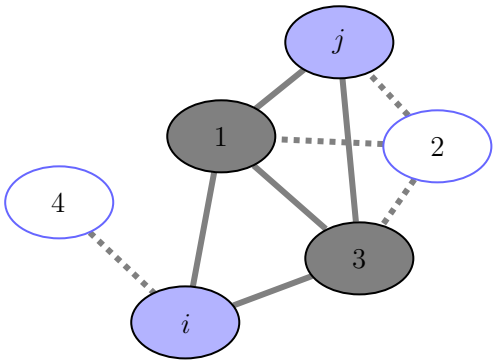


Abbildung 2.6.2: Allgemeines Beispiel zur Bestimmung von $\mathcal{C}_{ij}$ und $\bar{K}_{ij}$ [KC09, S. 177].

Abbildung 2.6.2 verdeutlicht das Prinzip der CND am Beispiel zweier Knoten i und j und eines daraus resultierenden Subgraphen G_{ij}, der durch die direkten Nachbarn $\{1, 3\}$ (grau ausgefüllt) mit den grau dargestellten Kanten aufgespannt wird. Für diese Knoten i und j lässt sich $\bar{K}_{ij}$ wie folgt bestimmen:

$\mathcal{C}_{ij}$ des Subgraphen G_{ij} umfasst nach (2.6.2) die Knotenmenge $\{i, j, 1, 3\}$. Somit erhält man für die Gewichtung nach (2.6.1), dass $\bar{K}_{ij} = 4^2 \cdot \frac{10}{12} = 13.\bar{3}$ (mit $\gamma = 2$) beträgt. Dies lässt auf eine hohe lokale Dichte und somit hoher Gemeinschaftszugehörigkeit von i und j schließen.

Beispiel 2.6.1 (Common Neighborhood Subgraph Density)**.** Wendet man dieses Verfahren auf den eingangs beschriebenen Beispielgraphen G_{bsp} an, so erhält man einen gewichteten Netzgraphen (G_{cnd}, w) gemäß Abbildung 2.6.3. Hierbei wurde $\bar{K}_{ij}$ für alle adjazenten Knoten i, j mit $e_{ij} \in E(G_{\text{bsp}})$ berechnet und den inzidierenden Kanten als Gewichtung zugewiesen. Dabei ist gut ersichtlich, dass Kanten, wie (c, d), deren Knoten offensichtlich nicht zur selben Gemeinschaft gehören, ein deutlich geringeres Gewicht $w(e_{cd})$ besitzen, als Kanten innerhalb von Gemeinschaften (beispielsweise $w(e_{ef})$).

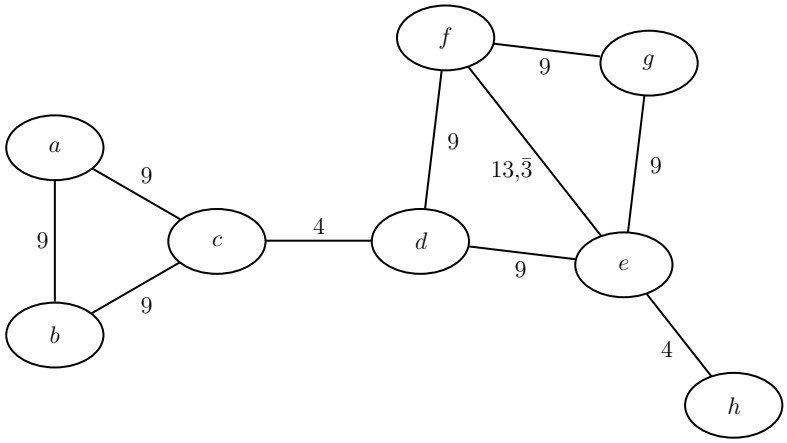


Abbildung 2.6.3: Beispielgraph (G_{cnd}, w) mit $\gamma = 2$ (entstanden aus (G_{u}, w))

2.7 Clustering-Verfahren

Die Idee des Clusterings beruht auf der Problemstellung Daten, entsprechend ihrer Bedeutungen zueinander, unterschiedlichen Gruppen zuzuordnen. Dabei ist das Ziel, untereinander sehr ähnliche Daten in möglichst kompakten Mengen zusammenzufassen und dabei zu vermeiden, dass sich sehr unterschiedliche Daten in derselben Menge befinden. Hierbei kann es sich um Knoten in einem beliebigen Graphen G, Datenpunkte im $\mathbb{R}^n$ oder allgemeine Datentypen handeln, auf die Metriken oder Ähnlichkeitsmaße anwendbar sind. In den folgenden Abschnitten sollen jedoch in erster Linie Verfahren zum Clustering von reellwertigen Datenpunkten sowie Netzgraphen vorgestellt werden, da sich weitere Datentypen durch geeignete Verfahren, wie sie beispielsweise in Abschnitt 2.3.5 oder Abschnitt 2.3.6 vorgestellt werden, in entsprechende reellwertige Datenpunkte oder Graphen mit Beziehungen, dargestellt durch Kanten, überführen lassen.

Datenpunkte Soll ein Clustering von Datenpunkten – meist auf Basis von reellwertigen Vektoren – durchgeführt werden, so müssen als erstes Ähnlichkeitsbeziehungen unter diesen berechenbar sein. Da in reellwertigen Vektorräumen Metriken definiert sind (vergleiche dazu Definition 2.3.2), können zwischen beliebigen Punkten euklidische Abstände berechnet und diese anschließend in Ähnlichkeitsmaße überführt werden (siehe (2.3.17) bis (2.3.19)).

Graphen Bei dem Clustering von Graphen hingegen soll für jeden Knoten entschieden werden zu welcher Gruppe er gehört. Dies geschieht auf Grundlage der Ähnlichkeitsbeziehungen, die durch die Kanten zwischen einzelnen Knoten ausgedrückt werden. Hierbei wird üblicherweise von einem *Netzgraphen* (siehe Definition 2.2.1) mit Kantengewichten ausgegangen.

Falls keine Informationen über die Kantengewichte vorliegen, kann eine Vorverarbeitung (siehe Abschnitt 2.6) durchgeführt werden, um die Zuverlässigkeit von Clustering-Algorithmen zu verbessern.

2.7.1 k-means

Handelt es sich um eine Menge reellwertiger Daten $x_1, \ldots, x_n \in \mathbb{R}^n$, die einzelnen *Clustern* zugeordnet werden sollen, so bietet sich als naiver Ansatz nach Ertel [Ert08, S. 227ff] der k-means-Algorithmus, insbesondere wenn „[...] *die Zahl der Cluster schon im Voraus bekannt ist* [...]“ [Ert08, S. 227], an. Hierbei werden zuerst k Prototypen $w_j \in \mathbb{R}^n,\ j \in \{1, \ldots, k\}$ festgelegt, die jeweils einen Cluster j repräsentieren sollen. Dabei können die Prototypen mit dem Wert eines existierenden Datenpunktes, oder zufällig initialisiert werden. Das Ziel dabei ist, dass jeder Datenpunkt x_i seinem nächstgelegenen Prototypen w_j zugeordnet wird. Die Menge der Prototyp w_j zugeordneten Daten entspricht dann dem Cluster j. Das Verfahren verläuft iterativ, indem in jeder Iteration die Prototypen in den Schwerpunkt ihrer Datenpunkte verschoben werden und anschließend erneut zu jedem Prototypen seine nächstgelegenen Datenpunkte bestimmt werden. Dies geschieht so lange, bis die Clusterzuordnung und die Position der Prototypen in einen stabilen Zustand konvergieren. Die Abstände zwischen Datenpunkten und Prototypen werden dabei mit Hilfe einer euklidischen Distanzfunktion gemäß Gleichung (2.3.10) bestimmt. Für den formalen Ablauf des Verfahrens siehe Algorithmus A.7.

2.7.2 LSH-Clustering: LSH-Link

Im Gegensatz zu Clustering-Verfahren wie k-means, in dem in jedem Schritt für jeden Datenpunkt entschieden wird, welchen Prototypen er am nächsten liegt und diesem dann zugeordnet wird, wird beim LSH-Clustering ein anderer Ansatz verfolgt. Das Verfahren als solches ist ein sogenanntes hierarchisches Clustering, welches von [KIW04] als **LSH-Link** bezeichnet wird. Dabei wird initial jeder Datenpunkt x als einzelner Cluster markiert. Anschließend werden alle Nachbarn von x, die sich innerhalb eines bestimmten Radius r befinden zu einem Cluster zusammengefasst. Dies geschieht iterativ mit sich vergrößerndem r solange, bis das Ergebnis einem gewünschten Clustering entspricht. Abbildung 2.7.1 stellt dabei die hierarchische Struktur dar, die bei einem solchen Verfahren zu beobachten ist: Erzeugt man, ausgehend von den einzelnen Datenpunkten, einen solchen Baum, der das Zusammenführen der einzelnen Cluster darstellt, so lässt sich in unterschiedliche Hierarchieebenen, von flach bis tief, unterscheiden. Ein optimales Clustering findet sich zwischen

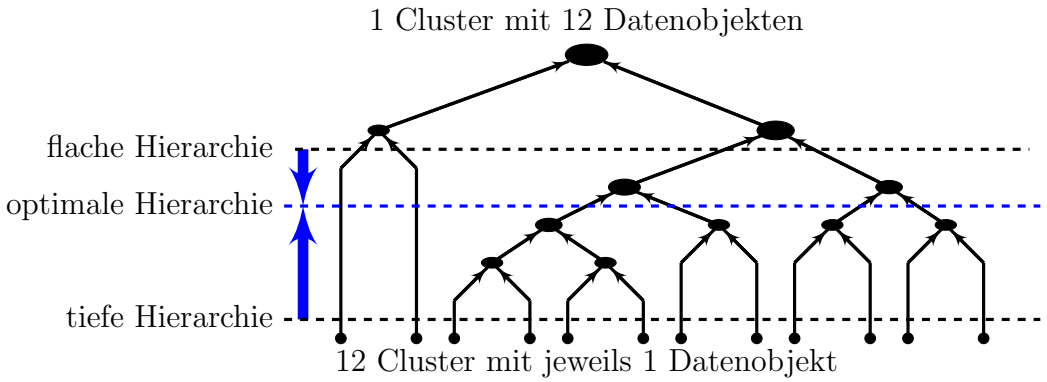


Abbildung 2.7.1: Hierarchisches Clustering (in Anlehnung an [KWZ98, S. 88])

beiden Extrema (n bzw. 1 Cluster) und kann sich, je nach Verteilung der Daten nach oben oder unten in der Hierarchie verschieben.

Damit nicht in jedem Schritt die Distanzen von x zu allen übrigen Punkten $x' \neq x$ berechnet werden müssen, bedient man sich, wie von [KIW04, S. 117ff] beschrieben, der Funktionalität eines leicht modifizierten *local sensitive hashings* (Abschnitt 2.4.2). So wird ein Punkt x nur dann einem Bucket B_h zugewiesen, wenn sich in diesem kein Punkt aus dem gleichen Cluster befindet, zu dem x gehört. Anschließend werden alle die Cluster zusammengefügt, zu denen sich Punkte in den gleichen Buckets befinden und die nicht weiter als r voneinander entfernt sind. Das Verfahren wird nun mit einem vergrößerten r wiederholt.

2.7.3 Affinity Propagation

Affinity Propagation (AfP) ist ein Clusteringverfahren für Datenpunkte. Dabei betrachtet es, im Gegensatz zu Verfahren wie k-means, jeden Datenpunkt als ein potentielles Exemplar, welche Zentren von Clustern repräsentieren. Nach Frey und Dueck [FD07a, S. 972] werden alle Datenpunkte als Knoten in einem Netzwerk angesehen, die iterativ entlang ihrer Kanten Nachrichten darüber austauschen, welche Knoten als Exemplare geeignet sind und welche zu diesen Exemplaren, mit anderen Worten Clustern, zugeordnet werden sollen.

Bei den zu übertragenden Nachrichten wird in „Verantwortlichkeiten“ (*responsibilities*) und „Verfügbarkeiten“ (*availabilities*) unterschieden. Die *responsibilities* $r(i,k)$, die von Datenpunkten i zu Exemplarkandidaten k verschickt werden, sagen dabei aus, wie gut k als Exemplar von i geeignet ist. *Availabilities* $a(i,k)$ hingegen, werden von den Kandidaten k aus an die Datenpunkte i übermittelt und teilen diesen mit, wie angebracht es für i wäre, k

als Exemplar zu wählen. Dabei werden für alle zu verschickenden Nachrichten r und a die zuvor Empfangenen berücksichtigt. Siehe hierzu Abbildung 2.7.2.

Beziehungen zwischen den Datenpunkten Für die Durchführung des Verfahrens ist es notwendig, dass zwischen den Datenpunkten Ähnlichkeiten $s(i,k)$ definiert sind. Diese lassen sich nach [FD07a] bei reellwertigen Vektoren beispielsweise durch den negativen *Euklidischen Abstand* zum Quadrat

$$s(i,k) = -||x_i - x_k||^2$$

abbilden. Somit entsteht ein Netzgraph (G, s) mit Ähnlichkeiten s als Kantengewichten. Bei den einzelnen Datenpunkten kann es sich somit entweder um reellwertige Vektoren, Knoten in einem bereits existierenden Netzgraphen (G, w), oder beliebige andere Datenobjekte handeln, zwischen denen Abstände oder Ähnlichkeiten berechenbar sind.

Ablauf des Verfahrens Das Vorgehen bei der AfP wird nach [FD07a] wie folgt beschrieben: Zu Beginn werden die *availabilities* zwischen allen Knoten i und k mit $a(i,k) = 0$ initialisiert. Anschließend werden $r(i,k)$ und $a(i,k)$ bezüglich jedes Knotens iterativ neu berechnet, indem in jedem Zyklus die Gleichungen

- $r(i,k) = s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + \mathrm{s}(i,k')\}$,
- $a(i,k) = \min \left\{0, r(k,k) + \sum_{i' \text{ s.t. } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\}$ für $i \neq k$ sowie
- $a(k,k) = \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(i',k)\}$

berechnet werden. Dies kann nach Aussage von Frey und Dueck [FD07a, S. 973] so lange geschehen, bis

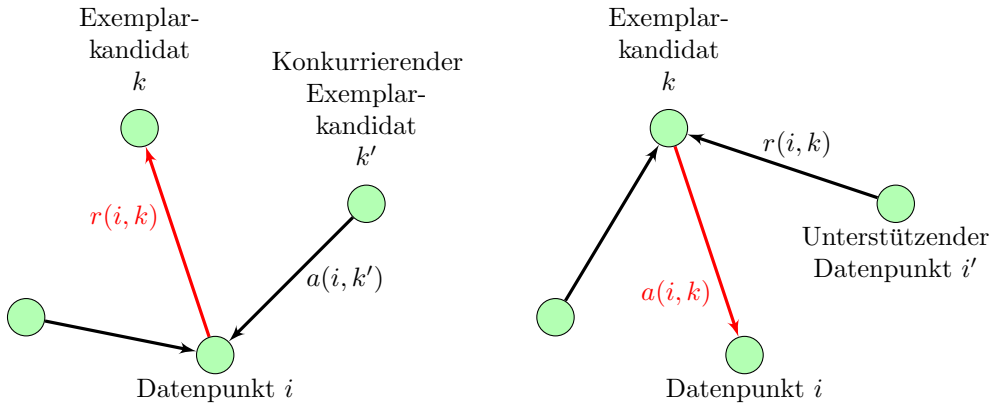
- die maximale Iterationszahl erreicht wurde,
- sich die Werte von $a(i,k)$ und $r(i,k)$ nur noch minimal ändern, oder
- die Entscheidungen bezüglich der Exemplare, sowie deren zugehörigen Datenpunkte hinreichend lange konstant bleiben.

Für eine formale Beschreibung des Ablaufes siehe Algorithmus A.8. Nach Abschluss des Verfahrens lassen sich in Anlehnung an [FD07a, S. 973] die Menge

I an Exemplaren k, sowie die Menge C_k an Knoten i, die einem Exemplar k zugeordnet werden durch

$$I = \{k \mid a(i,k) + r(i,k) \to \max!,\ i = k\}, \qquad \text{(Exemplarmenge)}$$
$$C_k = \{i \mid a(i,k) + r(i,k) \to \max!,\ i \neq k\} \qquad \text{(Clusterknotenmenge)}$$

ermitteln.



A) Senden von *responsibilities* **B) Senden von *availabilities***

Abbildung 2.7.2: Austausch von Nachrichten bei AfP [FD07a, S. 973]

Beispiel 2.7.1 (Affinity Propagation)**.** Ausgehend von dem in Abbildung 2.6.3 dargestellten Netzgraphen (G_{bsp}, w), führte die AfP nach insgesamt 10 Iterationen zu einem Clustering wie in Abbildung 2.7.3 zu sehen ist. Als Exemplare wurden durch den Algorithmus die Knoten c und e gewählt. Die gerichteten Kanten geben hierbei an, welchen Knoten k jeder Knoten i als Exemplar gewählt hat. Unter Berücksichtigung der Ähnlichkeiten zwischen Exemplaren und mit diesen adjazenten Knoten, führte dies zu einem Clustering in die Mengen $\{a, b, c\}$ und $\{d, e, f, g, h\}$.

2.7.4 Ant Colony Optimization

Clustering von Graphen mittels Ant Colony Optimization (ACO) funktioniert grundlegend anders als die bisher vorgestellten Verfahren. Liefen diese bisher streng deterministisch ab, d.h. eine bestimmte Eingabe führte abhängig von festgelegten Parametern stets zu dem selben Ergebnis, so ist bei der ACO dieses nicht garantiert, da es sich hierbei um ein stochastisches Verfahren handelt.

Der grundlegende Ablauf ist, dass ein Netzgraph G solange rekursiv in jeweils zwei disjunkte Subgraphen G' und G'' zerlegt werden soll (Abbildung 2.7.4), bis im letzten Rekursionsschritt ein optimales Clustering entsteht. Um eine solche Zerlegung von Graphen bewerten zu können, gibt es nach Mandala u. a.

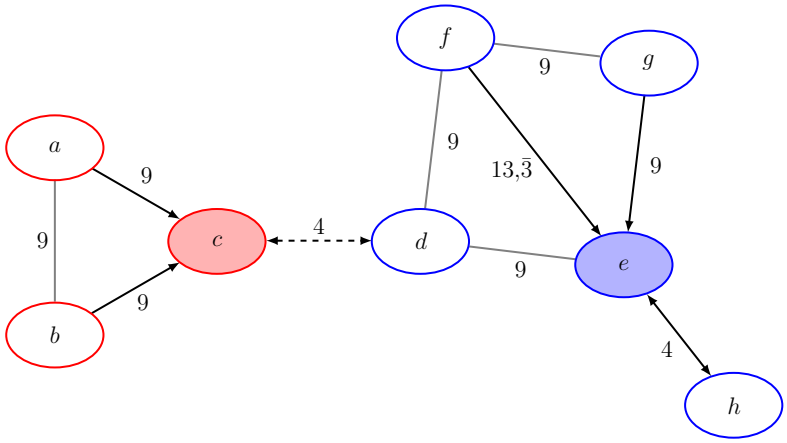


Abbildung 2.7.3: G_{afp}: Beispiel für AfP, angewendet auf G_{u}, bezüglich der Wahl von Exemplaren

[Man+11, S. 3] das Konzept einer *Modularitäts*funktion, die in Abhängigkeit eines zugrundeliegenden Netzgraphens, die Qualität der gewählten Unterteilung ermittelt.

Durch ein *Modularitätsmaß* $Q_G(x)$ [Man+11, S. 8] soll nun bestimmt werden, wie gut die Knoten des Netzwerks gruppiert sind, d.h. wie hoch die Qualität einer Partitionierung von G in zwei Subgraphen bezüglich eines Partitionsvektors $x \in \{0,1\}^n$ ist. Der Partitionsvektor gibt dabei die Gruppierung der Knoten $i, j \in V(G)$ zu $V(G') = \{i \mid x_i = 0\}$ und $V(G'') = \{j \mid x_j = 1\}$ an. Ist nun $x_i = x_j$, so folgt, dass i und j zum selben Subgraphen gehören.

Die Idee der ACO ist, dass jedem Knoten i eines Netzgraphen G ein Pheromonwert τ_i (initial $\tau_i = 0{,}5$) zugewiesen wird. Dieser drückt die Wahrscheinlichkeit aus, zu welchem der beiden künftigen Subgraphen der Knoten gehören soll. Im Verlauf des Verfahrens sollen, bildlich gesprochen, möglichst viele Ameisen **iterativ** einen Netzgraphen bewerten und anhand der Wahrscheinlichkeit, die durch τ_i ausgedrückt wird, die Knotenzugehörigkeit von i zu dem Subgraphen G' oder G'' festlegen. Die Knotenzugehörigkeit wird dabei durch den Lösungsvektor $s \in \{0,1\}^n$ festgehalten. Dabei durchläuft jede Ameise k in jeder Iteration zwei Phasen:

1. Gemäß der Wahrscheinlichkeitsverteilung τ wird eine gültige Lösung s_k bestimmt.

2. Anschließend wird, ausgehend von s_k, nach einer lokal optimalen Lösung s_k^* gesucht, indem von der Ausgangslösung alle n benachbarten Lösungen betrachtet werden, die sich lediglich um eine Stelle s_{ki} von s_k unterscheiden. Es wird also eine Lösungsmenge S_k aufgestellt, die ausschließlich Lösungsvektoren s'_k enthält, deren Hamming-Abstand (2.3.15)

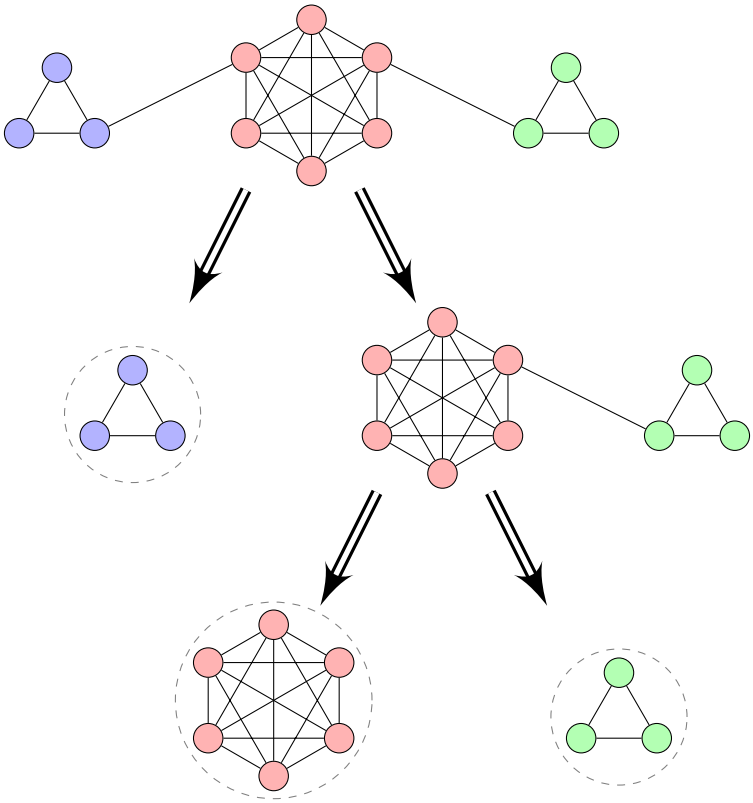


Abbildung 2.7.4: Rekursive Zerlegung eines Netzgraphens in drei Cluster mittels ACO [Man+11, S. 8]

$d_H(s_k, s'_k) = 1$ ist. Aus dieser Lösungsmenge wird nun eine lokal optimale durch

$$s^*_k = \max_{s'_k \in S_k}(Q_G(s'_k))$$

ermittelt, die einen maximalen Modularitätswert hat.

Haben alle k Ameisen eine für sich optimale Lösung s^*_k gefunden, so wird eine global optimale Lösung s^* aus den Lösungen der Ameisen, aus der bisherigen globalen, sowie aus der Lösung seit dem letzten Zurücksetzen von τ gebildet. Basierend auf s^* wird daraufhin eine Neubewertung von τ durchgeführt [Man+11, S. 10]. Ein Zurücksetzen von τ auf seinen Ausgangswert soll dabei laut den Autoren durchgeführt werden, falls τ bereits vor dem Ablauf der gewünschten Anzahl an Iterationen gegen eine optimale Lösung s^* konvergiert. Nach Abschluss der Iterationen wird der betrachtete Netzgraph G entsprechend dem Lösungsvektor x, der das bisher beste gefundene Modularitätsmaß besitzt, in die Zusammenhangskomponenten G' und G'' geteilt. Das bedeutet, dass alle Kanten, die mit Knoten aus G' und G'' inzidieren, entfernt werden.

Beispiel 2.7.2 (Ant Colony Optimization (ACO))**.** Im Folgenden soll das Verfahren der ACO auf den Beispielnetzgraphen G_{bsp} angewendet werden. Als Parameter wurden hierfür

- $|V(G)|$ Ameisen (bzw. $|V(G')|$ für jeden Subgraphen G'),
- 50 Iterationen,
- Konvergenzfaktor $\lambda = 0{,}99$,

festgelegt.

Die ACO führte im ersten Rekursionsschritt (Abbildung 2.7.5) zu einer optimalen Zerteilung von

$$s^*(G) = (0, 0, 0,\ 1, 1, 1, 1, 1),$$

aufgrund der Pheromonbewertung von

$$\tau^*(G) = (0{,}043; 0{,}043; 0{,}042;\ 0{,}961; 0{,}962; 0{,}962; 0{,}962; 0{,}962)$$

Durch den daraus resultierten Schnitt (grüne Trennlinie) entstanden die zwei Subgraphen

$$G' \text{ mit } V(G') = \{a, b, c\}$$

und

$$G'' \text{ mit } V(G'') = \{d, e, f, g, h\}.$$

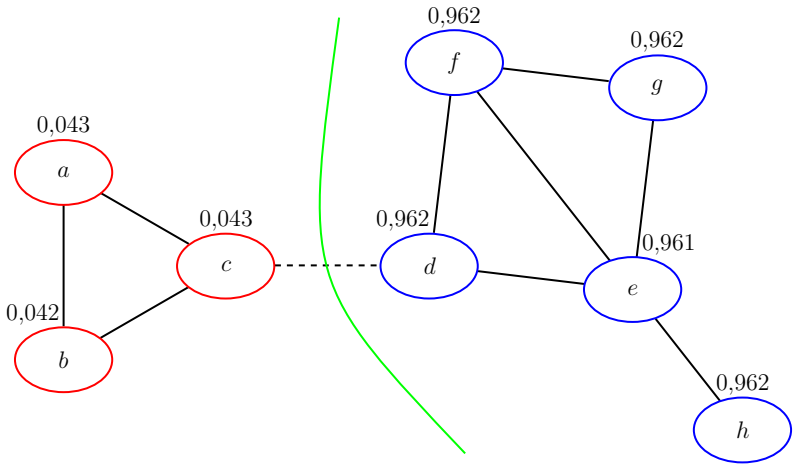


Abbildung 2.7.5: Eine beispielhafte ACO von G_{bsp} (erster Rekursionsschritt)

Im zweiten Rekursionsschritt wurde mit den gleichen Ausgangsparametern die ACO auf G' und G'' angewendet. Dabei wurden für G' insgesamt drei, sowie für G'' fünf Ameisen gewählt. Nach Ablauf der Iterationen kam es jedoch zu keiner weiteren Zerteilung, da aufgrund einer abschließenden Pheromonbewertung von

$$\tau^*(G') = (0{,}929; 0{,}928; 0{,}922)$$

und

$$\tau^*(G'') = (0{,}125; 0{,}05; 0{,}066; 0{,}067; 0{,}038)$$

eine Knotenzuordnung

$$s^*(G') = (1, 1, 1)$$

und

$$s^*(G'') = (0, 0, 0, 0, 0)$$

optimal war (siehe Abbildung 2.7.6). Der Grund hierfür liegt darin, dass der Knoten h nur einen Grad von 1 aufweist, wesshalb die Modularitätsfunktion eine weitere Zerteilung an dieser Stelle für ungünstig erachtet.

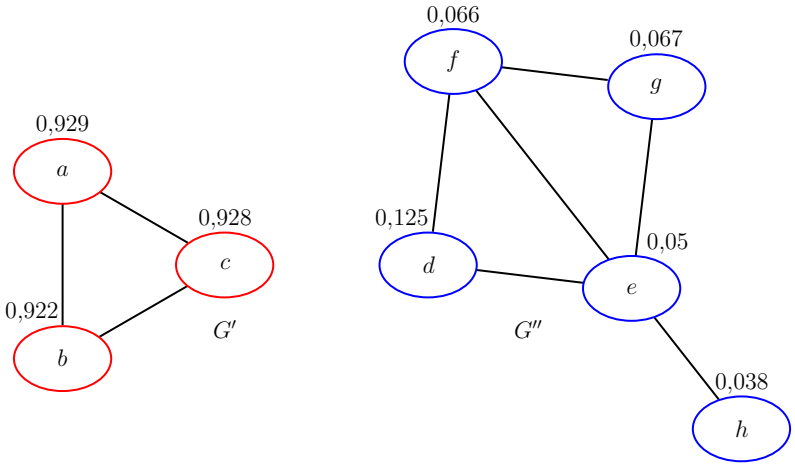


Abbildung 2.7.6: Beispielhafte ACO von G' und G'' (zweiter Rekursionsschritt)

3 Related Work

In diesem Abschnitt soll ein Überblick über Arbeiten gegeben werden, die mit den Themen dieser Arbeit verwandt sind. Hierbei soll auf Netzwerke eingegangen werden, in denen viele Teilnehmer mit Ressourcen vorhanden sind und sich in Gemeinschaften zusammenfinden. Darüber hinaus werden Arbeiten vorgestellt, die sich mit ähnlichkeitsbasierten Ansätzen in strukturierten P2P-Overlay-Netzwerken befassen.

3.1 OceanStore

Die OceanStore-Architektur (von Kubiatowicz u. a. [Kub+00]) wurde bereits von Stiefel [Sti10, S. 75ff] als ein Kandidat zur „*verteilte*[*n*] *Datenspeicherung*“ betrachtet. Sie soll eine sichere Verwaltung von Daten in nicht vertrauenswürdigen Umgebungen und eine hohe Verfügbarkeit sicherstellen [Kub+00, S. 2]. Aus diesem Grund würde sich OceanStore prinzipiell für den Einsatz in Netzwerken eignen, deren Teilnehmer sehr weit verteilt sind und auf gemeinsame Ressourcen zugreifen wollen (siehe Abbildung 1.1.2). Die Identifikation von Ressourcen findet hierbei jedoch ausschließlich mittels kryptographischen Hash-Funktionen wie SHA-1 (Secure Hash Algorithm 1) statt [Kub+00, S. 3].

Da jedoch ausgehend von einem Hash-Wert keine Rückschlüsse auf den Inhalt möglich sind (siehe auch Abschnitt 2.4.1) aber für die Erzeugung einer eindeutigen ID ein Verfahren wie SHA-1 unumgänglich ist, ist dieser Ansatz für eine *inhaltsbasierte Lokalisierung* (Abschnitt 4.2) ungeeignet.

3.2 Kademlia

Ein Verfahren zur ähnlichkeitsbasierten Ressourcenverwaltung in P2P-Netzwerken wurde von Maymounkov und Mazières [MM02] vorgestellt. Hierbei werden die mittels Distributed Hash Table (DHT) bereitgestellten Lookup-Informationen so verwaltet, dass ähnliche Ressourcen-IDs effizient gefunden werden können. Dieses wird erreicht indem die Hash-Werte der Ressourcen gemäß einer XOR-Metrik verglichen werden und benachbarte Teilnehmer für nahe beieinander liegende Ressourcen verantwortlich sind. Somit können für eine Anfrage nach einer bestimmten ID schnell alle ähnlichen lokalisiert werden.

Allerdings bezieht sich die XOR-Metrik bei Kademlia lediglich auf die Hash-Werte von Ressourcen, die wie bei OceanStore mittels SHA-1 berechnet werden. Aus dem gleichen Grund wie bei OceanStore ist somit dieser Ansatz für eine *inhaltsbasierte Lokalisierung* nicht geeignet.

3.3 Squid

Eine weitere Arbeit, die sich mit der ähnlichkeitsbasierten Verwaltung in P2P-Systemen befasst, ist die von Schmidt und Parashar [SP03]. Im Allgemeinen werden in P2P-Netzwerken Referenzen auf Ressourcen gleichverteilt. Dieser Umstand wirkt sich jedoch negativ auf die Lokalität der Daten aus.

Die Autoren entwerfen in ihrer Arbeit eine Erweiterung des Overlay-Netzwerks Chord mit dem Ziel *die Lokalität zu erhalten* [SP03, S. 228].

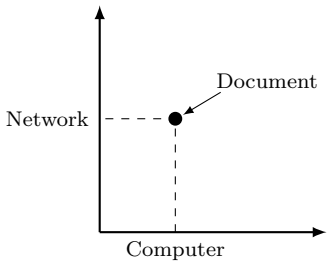


Abbildung 3.3.1: 2-Dimensionaler Schlüsselraum für ein Dokument [SP03, S. 229]

Hierbei kommen mehrdimensionale Schlüsselräume zum Einsatz, wie sie in Abbildung 3.3.1 dargestellt sind. Es ist nach Aussagen der Autoren möglich mit deren Erweiterung lexikographische Suchen durch den Einsatz von *Hilbert Space-Filling Curves (SFC)* nach benachbarten Ressourcen durchzuführen. Ein entsprechender Suchbaum für die Suche nach einem zweidimensionalen Schlüssel, dargestellt als 2-Tupel $(011, *)$ [SP03, S. 231] ist in Abbildung 3.3.2 visualisiert. Jeder Knoten in dem Baum stellt dabei ein Cluster mit ähnlichen Ressourcen bezüglich des rot dargestellten Präfixes der SFC-Werte dar.

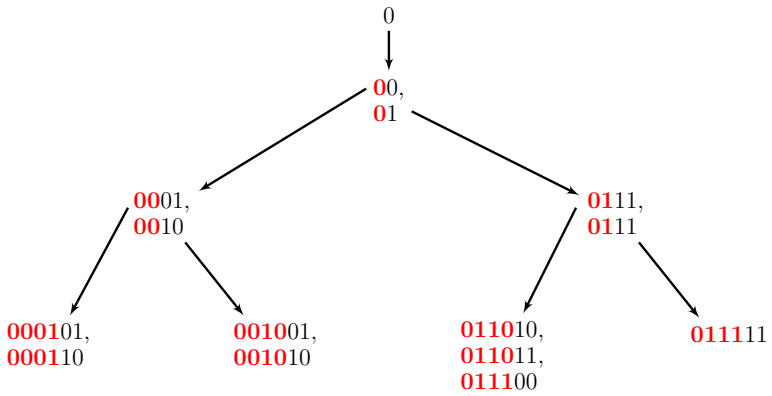


Abbildung 3.3.2: Rekursive Verfeinerung bei der Suche bezüglich eines *Queries* $(011, *)$. [SP03, S. 231], Präfixe hier rot hervorgehoben.

Shu u. a. [Shu+05] weisen in ihrer Arbeit jedoch darauf hin, dass Squid den Schlüsselraum statisch partitioniert (*„Squid [...] partition[s] the space statically“*) und dass der Partitionslevel im Voraus festgelegt werden muss (*„In*

Squid [...]*, the partitioning level needs to be decided beforehand.“*). [Shu+05, S. 2]

Im Rahmen dieser Arbeit wird hingegen ein Verfahren entworfen (Abschnitt 4.2.7, LSDM), bei dem keine statischen Partitionierungen entstehen, sondern vielmehr dynamisch die Schlüssel ähnlicher Ressourcen in sogenannten Buckets aufgeteilt werden. Dieser Ansatz eignet sich somit aufgrund der dynamischen Verfügbarkeit von unterschiedlichen Ressourcen in P2P-Systemen besser.

4 Ähnlichkeitsbasierte Verwaltung von Produktmodellen

In diesem Kapitel wird die Verwaltung von Produktdaten in dezentralen Netzwerkumgebungen behandelt. Gerade bei einer kollaborativen Produktentwicklung fallen Daten an, die für die beteiligten Entwickler verwaltet und diesen zur Verfügung gestellt werden müssen. Daher wird im Folgenden darauf eingegangen, welche Strategien für eine Datenverwaltung existieren und wie, insbesondere in dezentralen Kollaborationen, eine effiziente Bereitstellung von Produktmodellen aussehen kann.

4.1 Strategien zur effizienten Verwaltung von Produktdaten in Netzwerken

PDM-Systeme stehen, gerade bei einer hohen Komplexität der zu entwickelnden Produkte, vor der Aufgabe große Datenmengen zu verwalten und allen Kollaborationspartnern schnell und zuverlässig relevante Produktmodelle zur Verfügung zu stellen. Hierfür bieten sich unterschiedliche Ansätze im Bezug auf die zu verwendende Netzwerkarchitektur an. Im Folgenden soll daher ein Überblick über die „klassischen“ Ansätze der Datenverwaltung vermittelt werden und darüber hinaus deren Schwachpunkte bezüglich dem gewünschten Ziel dieser Arbeit aufgezeigt werden.

4.1.1 Klassische Ansätze

Client-Server-Netzwerke Die einfachste Art der Verwaltung von Produkten in Netzwerken stellt die Client-Server (CS) Architektur dar. Hierbei müssen Produktdaten, die den Netzwerkteilnehmern zur Verfügung gestellt werden sollen, auf einer oder mehreren Server-Instanzen abgelegt werden und können nur von diesen wieder abgerufen werden. Der Aufbau einer solchen CS-Architektur ist in Abbildung 4.1.1 schematisch dargestellt. Hierbei rufen Clients mittels *Requests* Daten vom Server ab, welcher in Form von *Responses* diese Daten übermittelt.

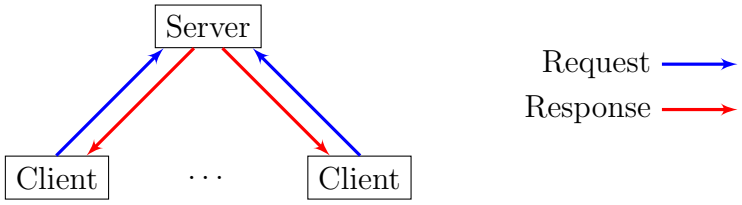


Abbildung 4.1.1: Aufbau der CS-Architektur

Der Vorteil bei einer solchen Architektur liegt darin, dass bei einer, wie in Abbildung 4.1.2 dargestellten, relativ zentralen Position des Servers eine hohe **Lokalität der Daten** gewährleistet werden kann. Dies bedeutet, dass Daten, um vom Server zum Client zu gelangen, nur relativ kurze Strecken zurücklegen müssen und somit sehr schnell verfügbar sein können.

Allerdings birgt die CS-Architektur auch Nachteile. Neben dem sofort ersichtlichen Nachteil, dass bei Ausfall des Servers die dort hinterlegten Daten nicht mehr verfügbar sind, existiert ein weiterer, auf den im Folgenden eingegangen wird. Um in einer solchen Netzwerkstruktur Daten ablegen zu können, müssen diese über mitunter mehrere Knotenpunkte übertragen werden, bis diese vom Server in Empfang genommen werden können. Bei dem Abrufen von Daten verhält es sich umgekehrt. Durch eine an den Server geschickte Anfrage, wird dieser veranlasst, Datenobjekte an den entsprechenden Teilnehmer (Client) zu übertragen.

Abbildung 4.1.2 zeigt die physikalische Sicht auf ein Netzwerk mit CS-Architektur. Dabei ist leicht ersichtlich, dass gerade im lokalen Umfeld um den

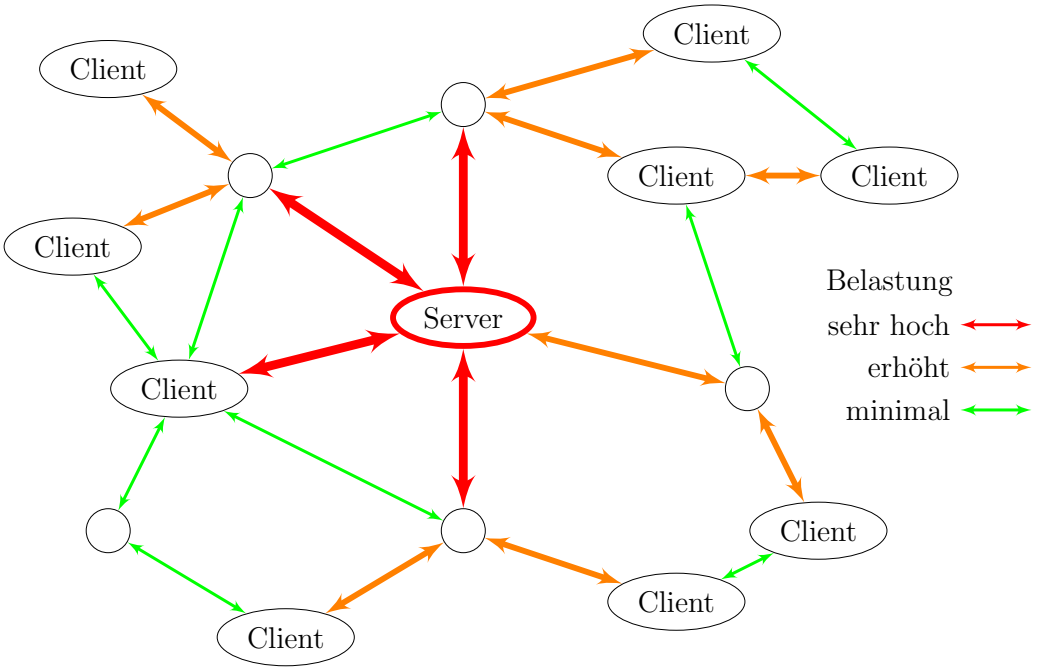


Abbildung 4.1.2: Hotspot bei der Client-Server-Architektur auf physikalischer Ebene

Server eine sehr hohe Last entsteht. Hierzu wurden in der Abbildung die Kanten orange bzw. rot eingefärbt, die voraussichtlich stark belastet werden, wenn die einzelnen Clients Daten mit dem Server austauschen. Dies hat zur Folge, dass, gerade in Netzwerken mit vielen Teilnehmern, eine sehr hohe Belastung der Datenleitungen um den Server zu erwarten ist. Somit kann es möglich sein, dass bei einer existierenden Beschränkung der verfügbaren Bandbreite, Daten

nur sehr langsam oder im schlimmsten Falle gar nicht mehr übertragen werden können, welches die **Hotspot-Problematik** der Architektur verdeutlicht. Ebenso tragen Ausfälle innerhalb der Netzwerkinfrastruktur zu der Gesamtproblematik bei. Haben Ausfälle von Clients so gut wie keine Auswirkungen, so führt ein Ausfall des zentralen Servers zu einem vollständigen Zusammenbruch der Architektur. Man spricht hierbei von einem sogenannten **Single Point of Failure**.

Betrachtet man die rein logischen Verbindungen zwischen Clients und Server, wie sie in Abbildung 4.1.3 dargestellt sind, so werden die zuvor beschriebenen Problematiken besonders deutlich. Die roten Pfeilmarkierungen stellen hierbei die Bereiche mit erwartungsgemäß hoher Netzlast sowie die durch einen Ausfall des Servers betroffenen Datenanbindungen dar.

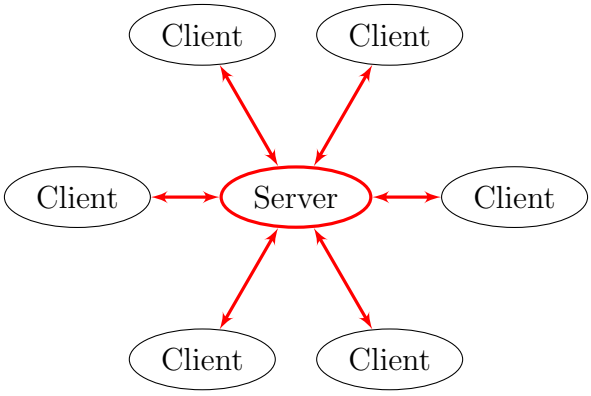


Abbildung 4.1.3: Hotspot bei der Client-Server-Architektur

Aus den oben genannten Gründen kann es daher sinnvoll sein, Strategien zu entwickeln, wie ein solcher Verlust an verfügbaren Ressourcen minimiert sowie die Belastung der Infrastruktur möglichst verteilt werden kann. Um dies zu erreichen, bietet sich die im folgenden Abschnitt beschriebene *Peer-to-Peer-Architektur* zur dezentralen Verwaltung von Ressourcen an.

Peer-to-Peer-Netzwerke Diese Art von Netzwerkarchitektur zeichnet sich dadurch aus, dass jeder Teilnehmer (*Peer*) die Rollen von sowohl Client als auch Server besitzt. Somit ist es möglich, Ressourcen nicht wie bei der Client-Server-Architektur zentralisiert, sondern verteilt zu verwalten. Hierbei unterscheidet man zwischen einer unstrukturierten P2P Architektur, in denen keine Ordnung der Teilnehmer vorgegeben ist, und einer strukturierten, in denen eine solche existiert.

In unstrukturierten P2P-Systemen (beispielsweise Gnutella) gestaltet sich die systematische Organisation von Datenobjekten als schwierig, da beispielsweise für Suchanfragen nicht entschieden werden kann, ob ein gesuchtes Datum

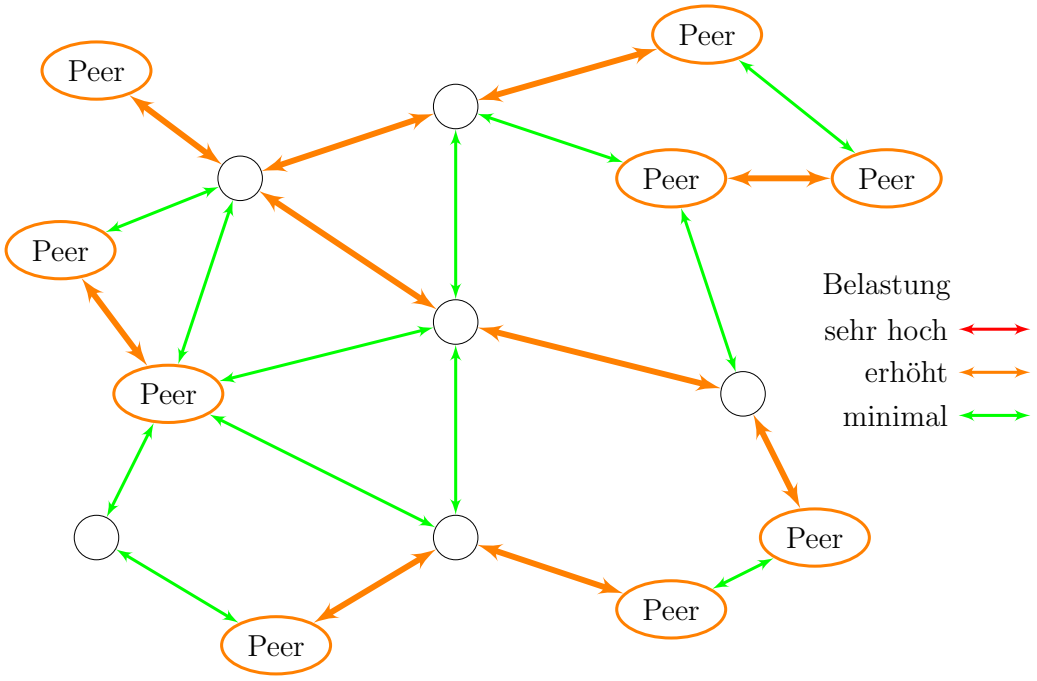


Abbildung 4.1.4: Hotspotbetrachtung der Peer-to-Peer-Architektur auf physikalischer Ebene

existiert oder nicht. Diese Konsequenz resultiert aus der Tatsache, dass Suchvorgänge, ausgehend von einem Teilnehmer, durch „Fluten“ des Netzwerkes durchgeführt werden. Die Reichweite solcher Suchanfragen wird dabei jedoch durch einen *Time-to-live*-Wert beschränkt. Somit sind unstrukturierte P2P-Netzwerke für eine zuverlässige Ressourcenverwaltung ungeeignet und sollen im Folgenden nicht weiter betrachtet werden.

In strukturierten P2P-Systemen (wie zum Beispiel FreePastry oder Chord) werden, Referenzen auf Ressourcen mittels Hash-Funktionen (beispielsweise SHA-1) auf die Teilnehmer verteilt. Hash-Funktionen haben dabei die Eigenschaft, dass sie beliebige Eingaben auf eine beschränkte Zielmenge abbilden. Eine solche Abbildung soll dabei möglichst gleichverteilt erfolgen. Das bedeutet, dass sich bei kleinen Änderungen der Eingabe die Ausgabe so stark ändert, sodass prinzipiell jedes Element des Zielraumes in Frage kommt. Im Regelfall werden jedoch, nach dem Prinzip der DHT, lediglich Referenzen auf Ressourcen verteilt. Somit wird die Netzwerkstruktur, bei einer Ressourcenverwaltung in der kein Einfluss auf den tatsächlichen Speicherort genommen wird, primär bei Suchvorgängen entlastet. Da die Ressourcen bei den Peers verbleiben, die diese in das System gestellt haben und nicht wie in der CS-Architektur zu einer zentralen Instanz übertragen werden müssen, erhöht sich die Ausfallsicherheit, da bei dem Ausfall eines Peers nicht sämtliche, im Netzwerk verfügbaren, Daten verloren gehen. Ein negativer Aspekt einer solchen gleichverteilten

Referenzverwaltung besteht in der geringen Lokalität der Daten. Damit ist gemeint, dass bei Anfragen nach bestimmten Ressourcen, aufgrund der Eigenschaften der zugrundeliegenden Hash-Funktionen, im Prinzip jeder Knoten des Netzwerkes hierfür verantwortlich sein kann. Aus diesem Grund kann es vorkommen, dass wenn sich ein für eine gesuchte Ressource verantwortlicher Peer in Bereichen mit vergleichsweisen hohen Latenzen befindet, eine schnelle Verfügbarkeit der Ressource eingeschränkt ist, da zwischen Suchanfrage und Lokalisierung der Ressource viel Zeit vergehen kann. Eine weitere Möglichkeit wäre es, jede Ressource r die von einem Peer p_i eingestellt wurde, direkt an einen geeigneten Peer p_j zu übergeben, für den $h(p_j) = h(r)$ gilt. Auf diese Weise würden alle Ressourcen gleichverteilt im Netzwerk vorliegen.

4.1.2 Peer-to-Peer-Anwendungsszenario: Branchenübergreifende kollaborative Produktentwicklung

Die Möglichkeiten einer dezentralen kollaborativen Produktentwicklung wurden, in Hinblick auf die Verwendung eines strukturierten Overlay-Netzwerkes, von Stiefel [Sti10] in seiner Dissertation bereits ausgiebig untersucht. Hierbei kam, zur Bereitstellung von Baugruppen bzw. -teilen und auch zur Unterstützung der Produktentwicklung, das P2P-System FreePastry (siehe hierzu [Dru+12] und [RD01]) als Backend zum Einsatz.

Stiefels Entscheidung, von einer bisherigen, allgemein zentralisierten Produktentwicklung, auf eine P2P-gestützte zu wechseln, bietet eine sinnvolle Möglichkeit zur Verteilung von Last und Verantwortlichkeiten. Für die Entwicklung von komplexen Baugruppen oder Produkten kann es zudem nützlich sein, wenn auf bereits durchgeführte Entwicklungsprojekte von Kollaborationspartnern zurückgegriffen werden kann. Ein solcher Ansatz wurde dabei von Stiefel noch nicht betrachtet, kann jedoch Vorteile im Entwicklungsprozess mit sich bringen, da beispielsweise unnötige Mehrfachentwicklungen vermieden werden können. Dieser Aspekt lässt sich darüber hinaus auch auf branchenübergreifende Kollaborationen erweitern. Hier könnte eine Suche nach Bauteilen oder Baugruppen, die zwar in nicht verwandten Projekten entwickelt wurden, aber unter Umständen für den betreffenden Entwicklungsprozess geeignet sind, vielversprechend sein.

Suche nach geeigneten Baugruppen Im Rahmen dieser Arbeit soll daher, im Prozess der verteilten kollaborativen Produktentwicklung, der Fokus auf der Lokalisation von bereits existierenden Baugruppen liegen. Mit der Lokalisation existierender Baugruppen ist gemeint, dass auch die Möglichkeit bestehen

soll, bereits existierende Produkte von Kollaborationspartnern ausfindig zu machen, die den benötigten Baugruppen hinreichend ähnlich sind. Bei einer solchen Suche in verteilten Produktmodellen ist auch ein branchenübergreifender Ansatz, beispielsweise bei der Verwendung identischer Motoren für Boote und Fahrzeuge, denkbar.

Beispiel: Entwicklung eines Zweizylindermotors Im Folgenden soll der zuvor beschriebene Anwendungsfall anhand der Entwicklung eines Zweizylindermotors näher erläutert werden. Wie der Abbildung 4.1.5 zu entnehmen ist, gliedert sich ein solcher Motor in mehrere, unterschiedliche Baugruppen bis hin zu einzelnen Bauteilen. Bei einer kollaborativen Produktentwicklung dieses

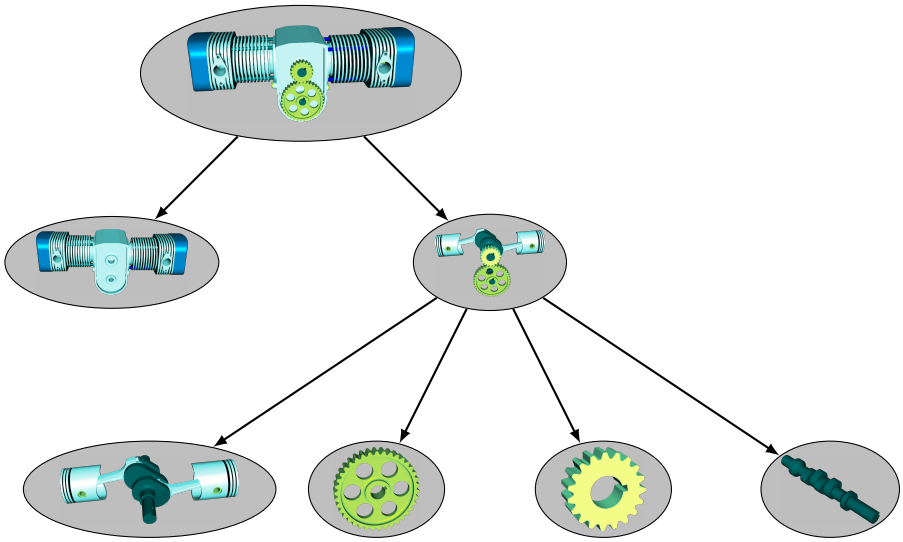


Abbildung 4.1.5: Dekomposition eines Zweizylindermotors (Baugruppen/Bauteile aus [Sie12a])

Motors kann es nun für den Projektinitiator von Interesse sein, ob es in dem für ihn zugänglichen Produktmodell bereits ähnliche Motoren aus früheren Entwicklungen gibt, die seinen Anforderungen entsprechen. Wäre dies der Fall, so könnte eine eigene Entwicklung eines solchen Motors eingespart werden. Aber auch wenn kein Fertigprodukt existiert, das dem gewünschten Exemplar ähnlich genug und somit kompatibel zu dem gewünschten Anwendungskontext ist, so kann es weiterhin von Interesse sein, ob kompatible Unterbaugruppen des Motors verfügbar sind. Auf diese Weise kann ein Verantwortlicher für die Marktsichtung nach Fertigprodukten den Motor gemäß Abbildung 4.1.5 in Unterbaugruppen zerlegen und nach diesen suchen. Für einen detaillierten Ablauf dieses Prozesses sei auf die Abbildung A.1 verwiesen. In dem dort abgebildeten Aktivitätsdiagramm sind die für diese Arbeit besonders relevanten Aktivitäten – das Lokalisieren von ähnlichen Ressourcen – *blau* hervorgehoben. Bei der *rot*

markierten Aktivität handelt es sich um eine Hinterlegung für einen, analog zu dem dargestellten Prozess ablaufenden, Sub-Prozess, der jedoch aus Gründen der Übersichtlichkeit weggelassen wurde.

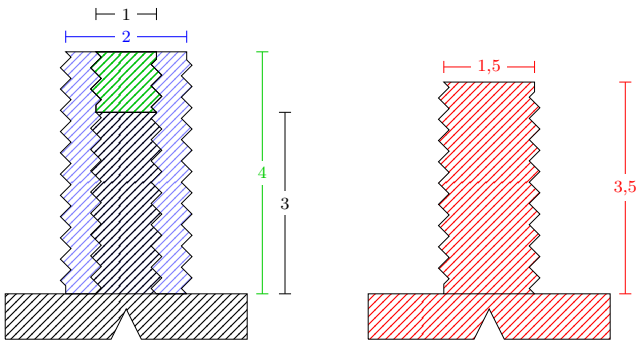
Um ähnliche Produkte, oder allgemein gesprochen *Ressourcen*, in verteilten Netzwerken lokalisieren zu können, muss es möglich sein, diese, gemäß ihren Ähnlichkeiten zueinander, zusammenzufassen, um so ähnliche Ressourcen effizient auszumachen. Als ein weiterer Aspekt bei der Suche ist dabei auch der Ausschluss nicht relevanter Ressourcen zu nennen, um bereits zu einem frühstmöglichen Zeitpunkt den relevanten Suchraum einschränken zu können.

4.1.3 Anforderungen an eine ähnlichkeitsbasierte Lokalisierung

Aufbauend auf dem zuvor geschilderten Anwendungsszenario lassen sich verallgemeinert Anforderungen ableiten, die für ein verteiltes Entwicklungssystem gelten müssen, wenn in diesen eine Suche und Lokalisierung ähnlicher Ressourcen möglich sein soll.

Üblicherweise werden in P2P-Systemen, wie in Abschnitt 4.1.1 beschrieben, Ressourcen nach ihrer ID indiziert und sind nur über diese abrufbar. Das bedeutet, dass zur Lokalisierung von Ressourcen entweder eine konkrete ID bekannt sein muss, oder schlicht sämtliche Ressourcen des Netzwerkes indiziert werden müssen, um eine Ressource mit den gewünschten Eigenschaften zu finden. Sollen jedoch Datenobjekte lokalisiert werden, die sich ähnlich bezüglich inhaltlicher Kriterien sind, so ist die ID einer Ressource als alleiniges Schlüsselwort zur Suche ungeeignet, da es in P2P-Systemen keinen Rückschluss auf den Inhalt erlaubt. Ebenso kann eine vollständige Indizierung der verfügbaren Ressourcen mit einem hohen Aufwand verbunden sein und müsste für jede individuelle Suche erneut durchgeführt werden. Sinnvoller gestaltet sich dabei der Ansatz, vor der eigentlichen Suche, bereits eine gewisse inhaltliche Vorstellung davon zu haben, wie die zu lokalisierende Ressource beschaffen sein soll. Es soll daher vorab eine *Referenzspezifikation* existieren, die den Suchkriterien entspricht, bezüglich derer ähnliche Ressourcen lokalisiert werden sollen.

Abbildung 4.1.6 zeigt dabei exemplarisch die Referenzspezifikation einer Schraube, wie sie als Anhaltspunkt dienen kann, ähnliche Bauteile zu finden. In dem konkreten Fall interessieren den suchenden Teilnehmer also alle Schrauben, mit 3 bis 4 Längeneinheiten (LE) und mit einem Durchmesser von 1 bis 2 LE (Abbildung 4.1.6 a). Im weiteren Verlauf dieser Arbeit werden allerdings für Suchanfragen konkrete Instanzen für ein Suchobjekt verwendet. Beispielsweise kann aus der dargestellten Spezifikation mit Toleranzen eine einzelne Instanz



a) mit Toleranzen **b) als „Durchschnitt“**

Abbildung 4.1.6: Referenzspezifikationen einer Schraube mit jeweils zwei Merkmalsausprägungen

einer Referenzspezifikation erzeugt werden, indem, wie in Abbildung 4.1.6 b) dargestellt, die Durchschnittswerte der Ausprägungen bestimmt werden.

Ein System, welches ähnlichkeitsbasierte Lokalisierungen ermöglichen soll, muss also Referenzspezifikationen entgegen nehmen und dem Suchenden Teilnehmer daraufhin Ressourcen-IDs übermitteln, die ähnlich zu den Suchkriterien sind. Diese IDs können dann zur Eingrenzung auf wenige, relevante Kandidaten genutzt werden.

4.2 Ähnlichkeitsbasierte Lokalisierung

Damit Produktdaten – oder verallgemeinert Datenobjekte – in Produktmodellen bezüglich ihrer inhaltlichen Ähnlichkeit zueinander gruppiert und entsprechend lokalisiert werden können, werden im Folgenden Voraussetzungen und Verfahren diskutiert, die notwendig sind, um dieses Ziel zu erreichen. Dabei sollen zunächst Verfahren zur ähnlichkeitsbasierten Gruppierung vorgestellt und anschließend dahingehend überprüft werden, ob sie den notwendigen Anforderungen aus Abschnitt 4.1.3 genügen. Anschließend folgen Entwürfe entsprechender Datenmodelle und Evaluationen bezüglich deren Anforderungen.

4.2.1 Metriken

Wie in Abschnitt 2.1 erläutert, sollen im Rahmen dieser Arbeit schwerpunktmäßig reellwertige Vektoren, die beispielsweise Merkmalsausprägungen von Bauteilen darstellen, sowie textuelle Beschreibungen von Spezifikationen relevant sein. Damit nun Relationen zwischen Datenobjekten gemessen werden können, müssen Metriken (siehe Abschnitt 2.3.1) existieren, die Distanzen bzw. Ähnlichkeiten ausdrücken können. Hierbei sei zudem zur Unterscheidung von Distanzen und Ähnlichkeiten auf Abschnitt 2.3.2 verwiesen.

Vektordarstellung von Merkmalsausprägungen Liegen Ressourcen mit klar definierten Merkmalsausprägungen vor, beispielsweise in Form von Schrauben, so können diese Ressourcen durch Vektoren dargestellt werden. Dieses kann

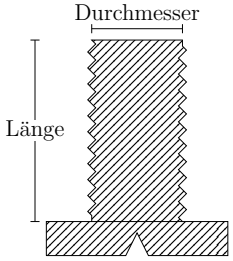


Abbildung 4.2.1: Merkmalsausprägungen einer Schraube

erreicht werden indem jedes Merkmal einem Vektorattribut entspricht. Somit erhält man bei n Merkmalen einen n-dimensionalen Vektor. Wurde für das zuvor genannte Beispiel festgelegt, dass jede Schraube genau zwei Merkmalsausprägungen besitzt (Länge und Gewindedurchmesser), so ergibt sich eine Vektordarstellung von:

$$v_{\text{Schraube}} = (\text{Länge}, \text{Durchmesser}).$$

In einer solchen Darstellung lassen sich Ressourcen bzw. hier Bauteile miteinander vergleichen, indem, wie in Abschnitt 2.3.3 beschrieben, Distanzen zwischen diesen berechnet werden. Hierbei bietet es sich an, einen *euklidischen* Abstand zu verwenden. Zudem kann es, bei besonders relevanten Merkmalen, mitunter auch sinnvoll sein, unterschiedlich hohe Gewichtungen bei der Distanzbestimmung einfließen zu lassen (Gleichung (2.3.16)).

Ein konkretes Beispiel für Metriken bezüglich Bauteilen ist in Abbildung 4.2.2 zu sehen. Es wurden hier drei Schrauben mit unterschiedlichen Merkmalen gegenüber gestellt. Die Abstände lassen sich, ausgehend von den Merkmalsvektoren

$$\begin{aligned} v_{\text{a}} &= (4{,}0; 2{,}0), \\ v_{\text{b}} &= (4{,}0; 1{,}0), \\ v_{\text{c}} &= (1{,}5; 2{,}0), \end{aligned}$$

mit Hilfe des euklidischen Abstandes (Gleichung (2.3.10)) berechnen als

$$\begin{aligned} d_e(v_{\text{a}}, v_{\text{b}}) &= d(v_{\text{b}}, v_{\text{a}}) = ||v_{\text{a}} - v_{\text{b}}|| = \sqrt{1} = 1, \\ d_e(v_{\text{a}}, v_{\text{c}}) &= d(v_{\text{c}}, v_{\text{a}}) = ||v_{\text{a}} - v_{\text{c}}|| = \sqrt{2{,}5} \approx 1{,}5811, \\ d_e(v_{\text{b}}, v_{\text{c}}) &= d(v_{\text{c}}, v_{\text{b}}) = ||v_{\text{b}} - v_{\text{c}}|| = \sqrt{3{,}5} \approx 1{,}8708. \end{aligned}$$

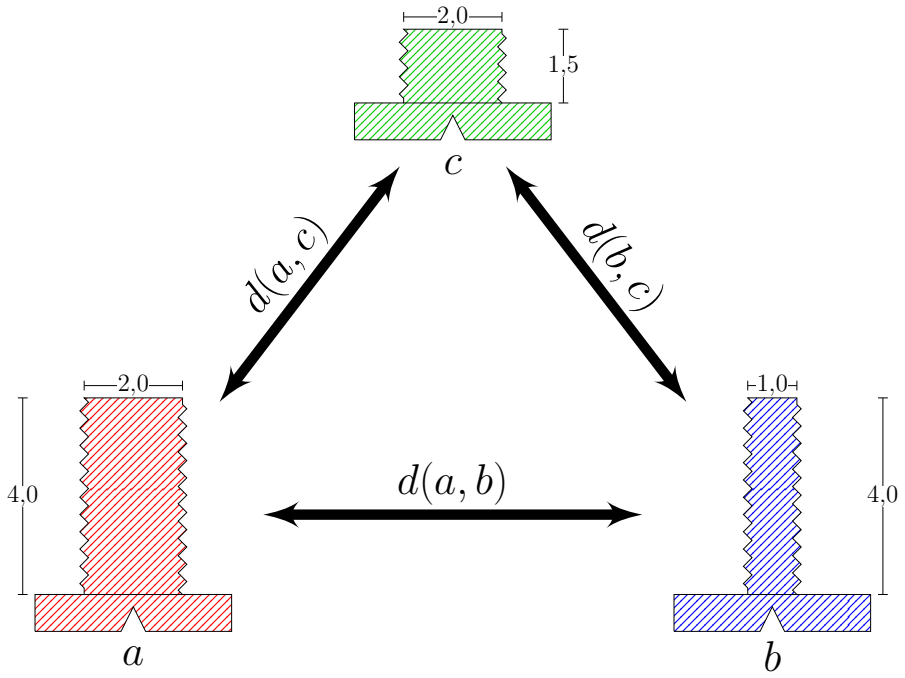


Abbildung 4.2.2: Distanzbestimmung zwischen Vektoren mit Merkmalsausprägungen am Beispiel von Schrauben.

Textuelle Daten Liegen hingegen die zu untersuchenden Ressourcen in Textform vor und können nicht in eine Vektordarstellung mit Merkmalsausprägungen überführt werden, so müssen die Texte auf inhaltlicher Basis direkt miteinander verglichen werden. Hierbei können Verfahren, wie *String kernel* sowie *Normalized Compression Distance* bzw. *Compression-based Dissimilarity Measure* (Abschnitt 2.3.5), eingesetzt werden.

Ausgehend von dem Schraubenbeispiel sähe eine textuelle Beschreibung der Schrauben wie in Abbildung 4.2.3 dargestellt aus.

Wird nun auf diese Texte das **String kernel**-Verfahren mit $\lambda = 0{,}5$ und $n = 4$ angewendet, so ergeben sich folgende Ähnlichkeiten:

$$\begin{aligned} K(\mathrm{a},\mathrm{b}) = K(\mathrm{b},\mathrm{a}) &\approx 0{,}0222, \\ K(\mathrm{a},\mathrm{c}) = K(\mathrm{c},\mathrm{a}) &\approx 0{,}0166 \text{ und} \\ K(\mathrm{b},\mathrm{c}) = K(\mathrm{c},\mathrm{b}) &\approx 0{,}0152. \end{aligned}$$

Ein weiteres Verfahren, welches in Abschnitt 2.3.5 vorgestellt wurde, ist die **Normalized Compression Distance**. Hier ergaben sich, unter der Verwen-

Spezifikation: Schraube *a*

```
Spezifikation der Schraube a
Laenge: vier komma null
Gewindedurchmesser: zwei komma null
```

Spezifikation: Schraube *b*

```
Spezifikation der Schraube b
Laenge: vier komma null
Gewindedurchmesser: eins komma null
```

Spezifikation: Schraube *c*

```
Spezifikation der Schraube c
Laenge: eins komma fuenf
Gewindedurchmesser: zwei komma null
```

Abbildung 4.2.3: Textuelle Spezifikation von Ressourcen am Beispiel von Schrauben.

dung des *bzip2*-Algorithmus (Kompressionslevel 9) zur Kompression der Texte, gemäß Gleichung (2.3.23) folgende Distanzwerte:

$$NCD(\mathrm{a},\mathrm{b}) = NCD(\mathrm{b},\mathrm{a}) = 0{,}175,$$
$$NCD(\mathrm{a},\mathrm{c}) = NCD(\mathrm{c},\mathrm{a}) = 0{,}224 \text{ und}$$
$$NCD(\mathrm{b},\mathrm{c}) = NCD(\mathrm{c},\mathrm{b}) = 0{,}24.$$

Zum besseren Vergleich mit den Ergebnissen der String kernel-Berechnung, wurde unter Verwendung der Gleichung (2.3.24) mit unveränderten Parametern das *Compression-based Dissimilarity Measure* bestimmt.

$$CDM(\mathrm{a},\mathrm{b}) = CDM(\mathrm{b},\mathrm{a}) \approx 0{,}5858,$$
$$CDM(\mathrm{a},\mathrm{c}) = CDM(\mathrm{c},\mathrm{a}) \approx 0{,}6041 \text{ und}$$
$$CDM(\mathrm{b},\mathrm{c}) = CDM(\mathrm{c},\mathrm{b}) \approx 0{,}6107.$$

Wie in der Abbildung 4.2.2 ersichtlich ist, sind sich Schraube *a* und *b* am ähnlichsten. Diese Beobachtung deckt sich mit den Ergebnissen der euklidischen Distanzen sowie der NCD und CDM.

4.2.2 Prototypen

Der Prototyp eines Datenclusters soll als *Identifikator* dienen, der bei Suchanfragen nach ähnlichen Datenobjekten genutzt werden kann. Er entscheidet darüber, ob infrage kommende Daten in dem von ihm repräsentierten Cluster

enthalten sind oder ob das betrachtete Cluster bereits frühzeitig ausgeschlossen werden kann.

Beispielsweise kann eine solche Menge ähnlicher Daten mit Prototypen gegeben sein, wie sie in Abbildung 4.2.4 mit drei getrennten Datenmengen (rot, grün und blau) dargestellt ist. Hierbei befinden sich die Prototypen der jeweili-

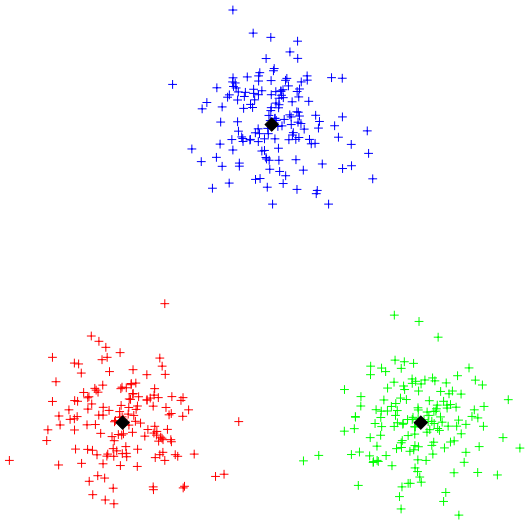


Abbildung 4.2.4: Datengruppen mit Prototypen (schwarz) im Schwerpunkt

gen Menge in deren Schwerpunkt und bilden so einen „Durchschnittswert“ dieser Datenobjekte. Bei einem Prototypen muss es sich nicht zwangsläufig um einen existierenden Datenpunkt, sondern kann sich unter Umständen auch um einen „virtuellen“ handeln. Ein solcher Fall ist in Abbildung 4.2.5 gut zu sehen,

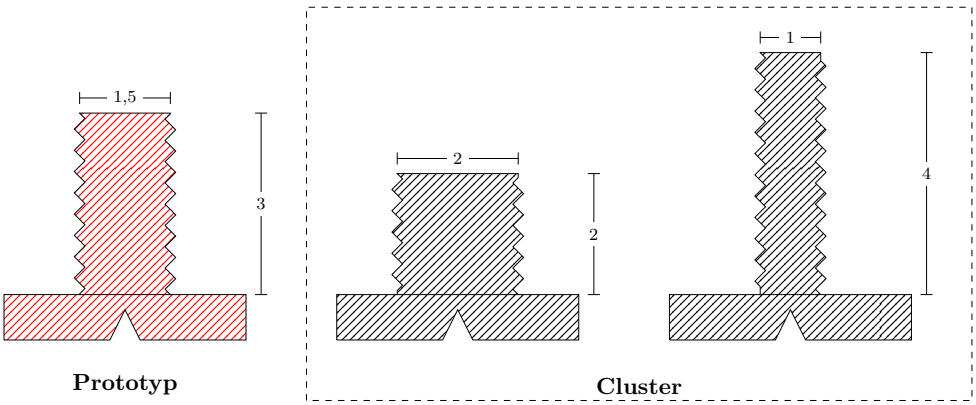


Abbildung 4.2.5: Prototyp (rot) bezüglich zwei ähnlicher Schrauben (schwarz) in einem Cluster

in dem lediglich zwei Schrauben in einem gemeinsamen Cluster existieren. Der Prototyp dieser Menge würde „zwischen“ den zwei real existierenden Schrau-

ben liegen und müsste somit künstlich eingefügt werden. Verallgemeinert ist daher für eine endliche Datenmenge, die beispielsweise in einem kontinuierlichen Vektorraum abgebildet wird, zu erwarten, dass sich in dem numerischen Schwerpunkt nicht zwangsläufig ein Datenpunkt befindet und somit ein „virtueller“ erzeugt werden muss, der diese Menge repräsentieren kann.

4.2.3 State of the Art: Clustering Verfahren

In diesem Abschnitt werden unterschiedliche Clusteringverfahren vorgestellt, die geeignet sind, Ressourcen nach ihrer Ähnlichkeit zu gruppieren. Es wird zudem gezeigt, wie mittels unterstützender Verfahren Daten, die in unterschiedlichen Formaten und Strukturen vorliegen, in Cluster gruppiert werden können.

Unterstützende Verfahren Die im Rahmen dieser Arbeit betrachteten, strukturierten Daten können in Textform, als Vektoren oder auch relative Ähnlichkeiten bzw. Distanzen vorliegen. Aus diesem Grund sind ggf. Methoden nötig, um die Daten für den Einsatz bestimmter Clustering-Verfahren aufzubereiten. Es existieren zudem Algorithmen, die ausschließlich auf Graphen operieren können. Da diese Eigenschaften im späteren Verlauf der Arbeit (Abschnitt 5.6.2) von besonderer Relevanz sind, ist neben der Überführung der Daten in *Ähnlichkeiten* bzw. *Distanzen* auch die Transformation in *gewichtete Netzgraphen* zu betrachten.

Im Folgenden wird daher auf Verfahren verwiesen werden, die geeignet sind, Umformungen von *strukturierten Daten* sowie *Netzgraphen* durchzuführen und in Kapitel 2 bereits erläutert wurden:

- **Strukturierte Daten** Sollen mittels Clustering textuell vorliegende Daten verarbeitet werden, so muss die paarweise Ähnlichkeit zwischen den Datenobjekten quantifiziert werden können. Zu diesem Zweck bieten sich die Verfahren der **String kernel** und **Normalized Compression Distance** an, wie sie in Abschnitt 2.3.5 vorgestellt wurden.

  Liegen hingegen Vektoren vor, beispielsweise Merkmalsausprägungen von Bauteilen, so können hieraus leicht Ähnlichkeiten mittels passender Metriken (Abschnitt 2.3.1) bestimmt werden. Ist jedoch eine Transformation in eine gewichtete Graphenstruktur notwendig, so lohnt sich der Einsatz von **Nearest-Neighboring**-Verfahren, wie sie in Abschnitt 2.5 erläutert wurden.

- **Ungewichtete Netzgraphen** Liegen keinerlei Kantengewichte vor, so lassen sich nur schwer Aussagen über den Grad der Beziehung zweier

Knoten treffen. Hierbei empfiehlt sich der Einsatz von Algorithmen, die, abhängig von der Vernetzungsdichte einzelner Knoten, sinnvolle Aussagen über einen Beziehungsgrad treffen können. Als ein solches Verfahren lässt sich die **Common Neighborhood Subgraph Density** (siehe Abschnitt 2.6.1) nennen.

Verfahren zum Clustering von strukturierten Daten

- **k-means** k-means ist ein bereits lange bekanntes Verfahren [Mac67], welches in der Lage ist reellwertige Daten im $\mathbb{R}^n$ prototypisch zu clustern. Dabei ist der Grundgedanke, wie in Abschnitt 2.7.1 beschrieben, dass Prototypen die Datenpunkte zugeordnet bekommen, denen sie am nächsten gelegen sind und iterativ in deren Schwerpunkte verschoben werden (siehe auch Abbildung 4.2.4). Veranschaulicht ausgedrückt können sie, wie in dem Beispiel mit den Merkmalsausprägungen von Schrauben (Abbildung 4.2.5), einen „Durchschnittswert“ von diesen darstellen. Das Verfahren wurde seit seiner ersten Publikation deutlich weiterentwickelt. Hier ist beispielsweise **Relational Neural Gas** [HH07] zu nennen, welches von Hammer und Hasenfuss publiziert wurde.

- **Affinity Propagation** Wie in Abschnitt 2.7.3 beschrieben, handelt es sich hierbei um ein graphenbasiertes Clustering. Das bedeutet, dass vorliegende Datenpunkte zuerst, durch die in Abschnitt 2.5 erläuterten Methoden, in eine Graphenstruktur überführt werden müssen. Bei der AfP können die einzelnen Knoten als untereinander verbundene Peers in einem Netzwerk verstanden werden, die Nachrichten austauschen. Hierbei werden zwei Arten von Nachrichten ausgetauscht. Zum einen wie sehr ein „Peer“ seinen Nachbarn als geeigneten Repräsentanten seines Clusters sieht und zum anderen wie gut er sich selbst als Repräsentant sehen würde. Im Laufe des Verfahrens entsteht hierbei ein Gleichgewicht, indem sich einzelne Cluster mit Repräsentanten herausbilden.

  Darüber hinaus kann nach [KC09, S. 178] die AfP zur Identifikation von Gemeinschaften in ungewichteten Netzgraphen eingesetzt werden, wenn für diese zuvor die *Common Neighborhood Subgraph Density* bestimmt wurde.

- **Ant Colony Optimization** Einen interessanten Ansatz verfolgt das graphenorientierte Verfahren der ACO, da es gegenüber den anderen hier vorgestellten Verfahren den Vorteil hat, dass es auf ungewichteten Netzgraphen operieren kann. Der Grundablauf, wie in Abschnitt 2.7.4 detail-

liert erläutert, gleicht dabei einem hierarchischen Clustering, d.h. es wird rekursiv eine Zweiteilung (*bisection*) eines (Teil-)Graphens in Untergraphen durchgeführt. Hierbei kommt eine Kostenfunktion zum tragen, die entscheidet, welche Zerteilung zu bevorzugen ist und wann das Verfahren terminiert. Um eine *bisection* durchführen zu können, sollen Ameisen mittels Pheromonspuren vielversprechende „Kandidaten“ mit minimalen Kosten finden. Nach einer gewissen Anzahl an Iterationen bildet sich dabei eine Präferenz heraus, gemäß dieser der Graph dann getrennt wird.

- **LSH-Link** Bei dem LSH-basierten Clustering wird ähnlich wie bei der ACO ein hierarchischer Ansatz verfolgt. Bei der Umsetzung wird allerdings von einzelnen Datenpunkte ausgegangen, die dann schrittweise zu größeren Clustern zusammengesetzt werden (siehe Abschnitt 2.7.2).

- **Spectral Clustering** Ein weiteres Verfahren aus der Familie der Graphen basierten Clustering-Verfahren ist *Spectral Clustering*. Hierbei können ebenfalls Datenpunkte verwendet werden, wenn diese zuvor in einen gewichteten Graphen überführt wurden. Anschließend wird versucht möglichst minimale Schnittkosten bei der Zerteilung des Graphens zu verursachen. Das Verfahren lässt sich daher nach [Lux07, S. 401] auf das *Graph-Cut*-Problem zurückführen.

In der Abbildung 4.2.6 sind die untersuchten Clustering- und unterstützenden Verfahren in einer gemeinsamen Übersicht dargestellt. Hierbei wird zudem

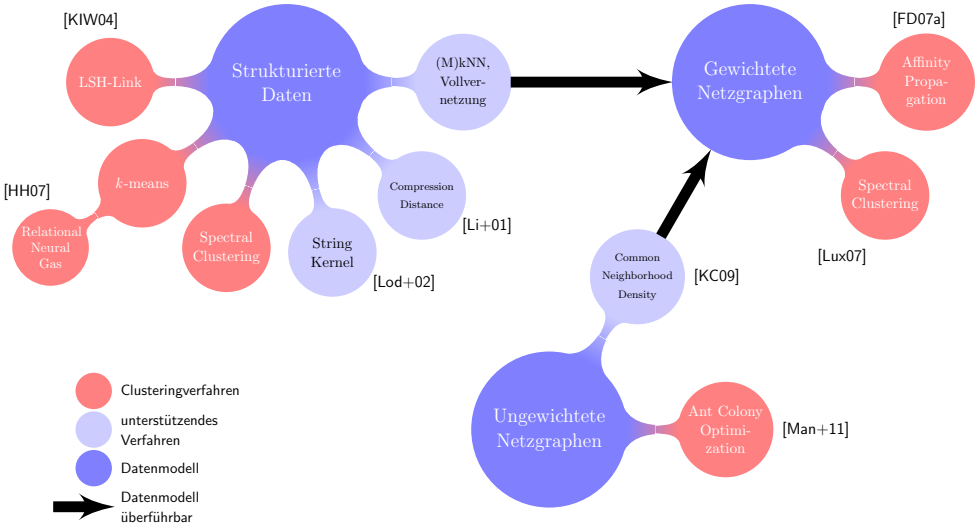


Abbildung 4.2.6: Einordnung der relevanten Clustering-Verfahren

deutlich, dass für jede der betrachteten Datenstrukturen mindestens ein Verfahren existiert, welches die entprechende Struktur clustern kann und darüber

hinaus, dass eine Überführung in *gewichtete Netzgraphen* möglich ist. Eine solche Überführung ist ein Aspekt, der bei einer späteren Umsetzung der im Rahmen dieser Arbeit erstellten Datenmodelle und Wahl der Clustering-Verfahren, berücksichtigt werden sollte.

4.2.4 Clustering Data Model

Das Clustering Data Model (CLDM), welches in diesem Abschnitt vorgestellt wird, soll die zuvor beschriebenen Anforderungen einer ähnlichkeitsbasierten Suche innerhalb von verteilten Systemen umsetzen (Abschnitt 4.1.3). Hierbei ist es erforderlich, dass die notwendigen Anforderungen spezifiziert werden, nach denen ein solches System erstellt werden kann.

Anforderungen an das CLDM Bezüglich des zu entwerfenden Datenmodells wird gefordert, dass

1. alle im Netzwerk bereitgestellten Ressourcen, gemäß ihrer paarweisen Ähnlichkeiten, gruppiert werden können,
2. eine Suche nach ähnlichen Ressourcen mittels Referenzspezifikationen und Prototypen möglich ist sowie
3. das Hinzufügen, Modifizieren und Entfernen von Ressourcen unterstützt wird.

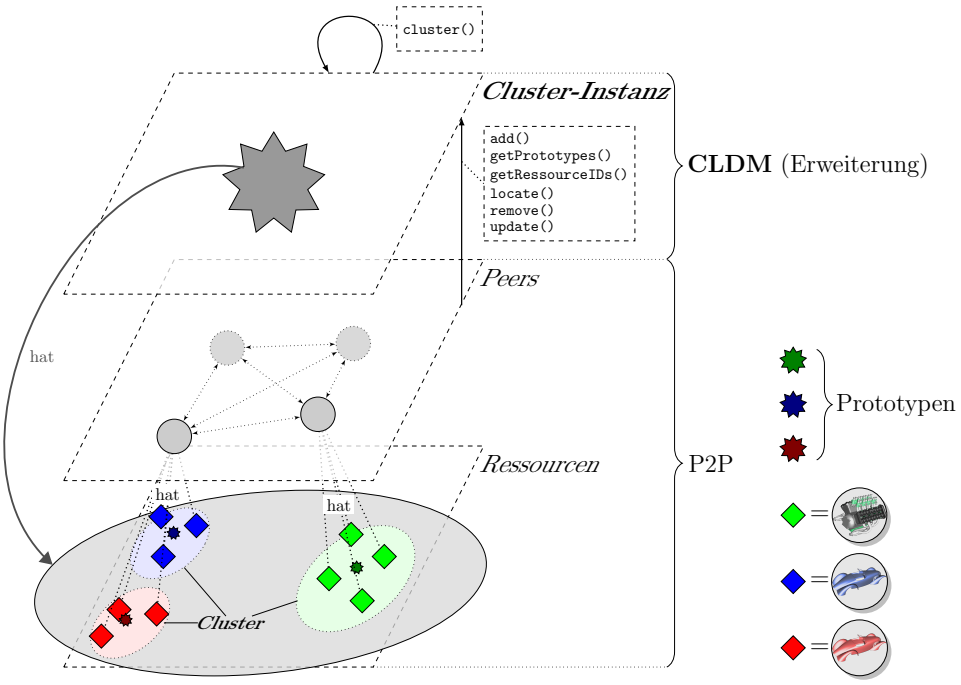


Abbildung 4.2.7: CLDM Architektur (Baugruppen aus [Sie12b])

Das CLDM soll, wie in Abbildung 4.2.7 dargestellt, als eine 3-Schichten-Architektur, bestehend aus *Ressourcen-*, *Peer-* sowie *Cluster-Instanz*-Schicht (Cluster-Instanz (CI)), modelliert werden. Die beiden untersten Schichten können durch ein beliebiges P2P-System realisiert werden, sofern dieses in der Lage ist, Ressourcen anhand einer eindeutigen ID den einzelnen Peers zur Verfügung zu stellen. Hierbei soll es für die Gesamtsystemarchitektur unerheblich sein, ob die Daten mittels DHT verwaltet werden oder bei dem verantwortlichen Knoten physikalisch vorliegen. Es soll ferner davon ausgegangen werden, dass eine Anfrage nach einer bestimmten Ressourcen-ID – sofern diese im Netz existiert – stets korrekt vom System beantwortet wird, d.h. dass die Ressource zum anfragenden Peer übertragen wird.

Eine darüber hinausgehende Betrachtung der konkreten Ressourcenverwaltung in der P2P-Schicht soll im Rahmen dieser Arbeit nicht geschehen. Die Gründe hierfür liegen in der **Vermeidung unnötiger Komplexität** sowie einer gewünschten **Unabhängigkeit** gegenüber des zugrundeliegenden P2P-Backends. Es wird somit zusammenfassend als Grundvoraussetzungen getroffen, dass

1. eine eindeutige Identifikation der Ressourcen sowie
2. eine garantierte Übertragung der geforderten Ressourcen

gegeben ist.

Die Realisierung der CLDM soll durch den Einsatz einer zentralen *Cluster-Instanz* geschehen. Diese CI empfängt, wie in Abbildung 4.2.7 dargestellt, die Ressourcen aller Teilnehmer (`add()`) und teilt diese in Cluster auf (`cluster()`). Hierbei soll ein **prototypenbasiertes** Clustering eingesetzt werden, um die Suche nach relevanten Clustern ermöglichen zu können. Das Ziel ist dabei, Referenzen von erzeugten Prototypen auf IDs der dazugehörigen Ressourcen abzubilden. Stellen nun Teilnehmer ihre Anfragen mittels *Referenzspezifikationen* von gesuchten Ressourcentypen, so bestimmt die CI die Prototypen, die zu der Anfrage am ähnlichsten sind. Über die infrage kommenden Prototypen lassen sich nun effizient die Cluster bestimmen, in denen Ressourcen zu finden sind, die für den Teilnehmer von Interesse sein könnten. Im letzten Schritt kann die Cluster-Instanz die relevanten IDs an den suchenden Teilnehmer übertragen, welcher diese mithilfe der Funktionalität des P2P-Backends beziehen kann.

Im Folgenden soll auf den Ablauf der für die CLDM vorgesehenen Operationen in Tabelle 4.2.2 im Detail eingegangen werden. Es sei an dieser Stelle im Besonderen auf die **Lokalisierung anhand von Referenzspezifikationen** verwiesen, da dieser Vorgang die Kernfunktionalität des Datenmodells

ausmacht. Hierbei werden zwei Varianten vorgestellt, wie ein Teilnehmer mit der CI interagieren muss, um ähnlich Ressourcen in dem Netzwerk zu finden.

Einheiten und Mengen, die für die Umsetzung notwendig sind, werden zum besseren Verständnis in Tabelle 4.2.1 näher erläutert.

Ressourcen (-mengen)	
r	allgemeine Bezeichnung einer Ressource
r_{new}	Ergebnis einer Ressourcenmanipulation
r_{old}	Ausgangspunkt einer Ressourcenmanipulation
R	Menge von Ressourcen

Clusteringspezifische Einheiten/Mengen	
q	Referenzspezifikation einer Suchanfrage
C_i	Mengen ähnlicher Ressourcen (Cluster)
$\texttt{ID}^q_{sim}$	Menge von IDs deren Ressourcen ähnlich zur Referenzspezifikation q sind
P_r	Menge von Prototypen, die ähnlich zu einer Referenzspezifikation r sind

Tabelle 4.2.1: Relevante Bezeichnungen für das Clustering Data Model

CLDM Operationen (Peer→CI)	
`add(`R`)`	Füge der CI eine Menge R von Ressourcen hinzu.
`cluster()`	Führe ein prototypenbasiertes Clustering der Ressourcen durch. Hierbei werden die Cluster-Mengen C_i und die Prototypen P erzeugt.
`locate(`q`)`	Lokalisiere alle Prototypen aus P, die ähnlich zur Anfrage q sind. Gib anschließend die hiermit assoziierten ID^q_{sim} zurück.
`getPrototypes()`	Liefere alle Prototypen aus P an den anfragenden Peer zurück.
`getRessourceIDs(`P_q`)`	Liefere die IDs der Ressourcen zurück, die mit den Prototypen aus P_q assoziiert sind.
`update(`$r_{\text{old}}, r_{\text{new}}$`)`	Weise die CI an, dass die Ressource r_{old} durch r_{new} ersetzt werden muss.
`remove(`r`)`	Weise die CI an, dass die Ressource r entfernt werden muss.

P2P-Backend Operationen	
`addP2P(`r`)`	Füge Ressource r dem Netzwerk hinzu.
`updateP2P(`$r_{\text{old}}, r_{\text{new}}$`)`	Ersetze Ressource r_{old} durch r_{new}.
`removeP2P(`r`)`	Entferne Ressource r aus dem Netzwerk.

Tabelle 4.2.2: Vorgesehene Operationen für das Clustering Data Model

- **Übergabe von Ressourcen** Sollen dem Datenmodell Ressourcen übergeben werden, sprich ein Teilnehmer veröffentlicht diese in dem P2P-Netzwerk, so muss dieser Umstand auch der CI mitgeteilt werden. Der Ablauf des Verfahrens ist dabei in der Abbildung 4.2.8 in 3 Schritten

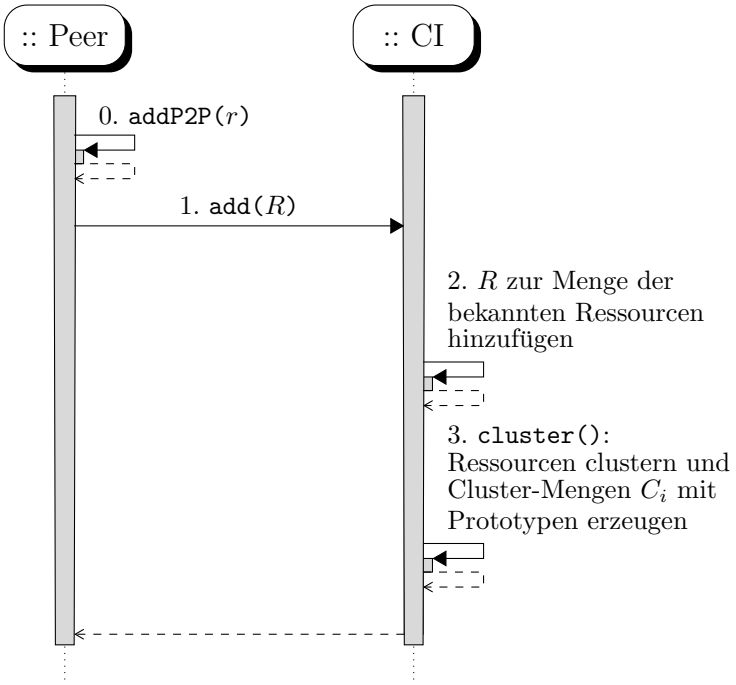


Abbildung 4.2.8: Ablauf der `add()`-Operation im CLDM bezüglich einer Ressourcenmenge R

dargestellt, wobei das Veröffentlichen der Daten im P2P als Schritt 0 gekennzeichnet ist. Zunächst muss hierbei ein Teilnehmer mittels `add()` die Menge seiner neuen Ressourcen an die CI übertragen (Schritt 1). Diese nimmt die Ressourcen entgegen (Schritt 2) und führt ein Clustering durch, wodurch alle Datenobjekte, die der CI bekannt sind, auf die Mengen C_i bezüglich ihrer Ähnlichkeit aufgeteilt werden. Darüber hinaus wird für jeden Cluster ein Prototyp erzeugt, der in dessen Schwerpunkt liegt und im späteren Verlauf der effizienten Identifikation relevanter Cluster dienen soll. Um eine solche Identifikation anhand von Prototypen zu ermöglichen, verknüpft die CI einen Prototypen mit den IDs der Ressourcen, die in dem Cluster liegen, für die der Prototyp „verantwortlich“ ist (`Prototyp` $\rightarrow \{$`Ressource`$, \ldots\}$).

- **Lokalisierung anhand von Referenzspezifikationen** Die Lokalisierung ähnlicher Ressourcen soll das Kernstück des CLDM bilden. Hierbei sollen effizient Ressourcen-IDs bestimmt werden können, die bezüglich der Referenzspezifikation einer Anfrage möglichst ähnlich sind. Wie Abbildung 4.2.9 (Variante A) zu entnehmen ist, geschieht dies in 4 Schritten. Der suchende Teilnehmer erstellt zuerst eine Referenzspezifikation, die als Ausgangspunkt seiner Suche dienen soll. Er fragt nun mittels `locate()` bei der CI an, welche Ressourcen bekannt sind, die seiner Spezifikation ähneln. Die Cluster-Instanz muss nun nichts weiter tun, als alle

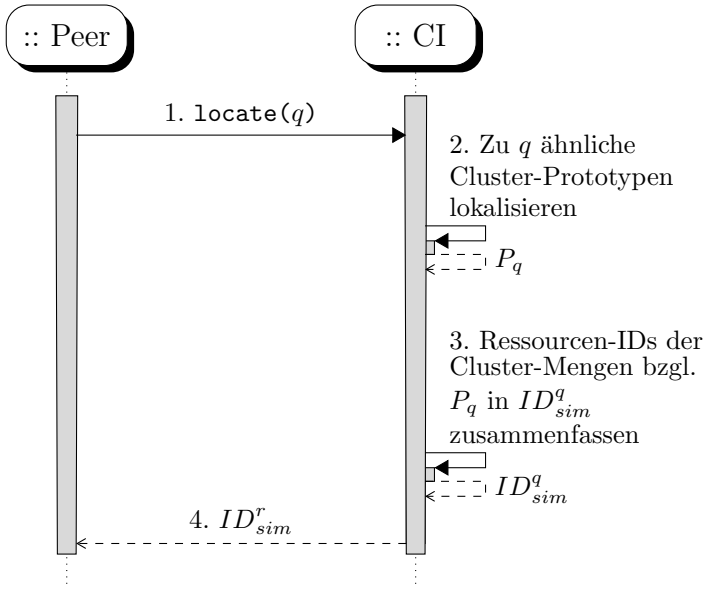


Abbildung 4.2.9: Ablauf der `locate()`-Operation im CLDM bezüglich einer Referenzspezifikation q (Variante A)

ihre Prototypen mit der Referenzspezifikation des Teilnehmers zu vergleichen, um relevante Ressourcen zu identifizieren und nicht relevante auszuschließen. Im letzten Schritt überträgt die CI alle Ressourcen-IDs, die in den relevanten Clustern liegen.

An dieser Stelle wäre allerdings auch ein weiterer Ansatz denkbar: Die CI könnte in Schritt 2 einfach sämtliche Prototypen an den Teilnehmer weiterreichen, damit dieser für sich Vergleiche anstellen kann und in einer weiteren Anfrage die für ihn interessanten Ressourcen-IDs von der CI bezieht. Dieser Fall ist in Abbildung 4.2.10 (Variante B) dargestellt.

- **Modifikation von Ressourcen** Ein ebenfalls wichtiger Aspekt stellt die Modifikation von Ressourcen dar. Wurden Ressourcen im P2P-Backend modifiziert, so muss dieser Umstand auch der Cluster-Instanz mitgeteilt werden (Abbildung 4.2.11). Die Benachrichtigung wird durch eine `update()`-Operation ausgelöst, wodurch der CI die alte und neue Ressource mitgeteilt wird. Mit einem Überschreiben des veralteten Datenobjektes allein ist es jedoch noch nicht getan. Je nachdem wie signifikant die Änderungen waren, kann es sein, dass die neue Ressource einem anderen Cluster zugeordnet werden muss und sich deren Prototypen ebenfalls ändern. Aus diesem Grund muss erneut ein Clustering durchgeführt werden.

- **Entfernen von Ressourcen** Werden Ressourcen aus dem P2P-Netz entfernt, so ist diese Information auch für die CI relevant. Denn durch

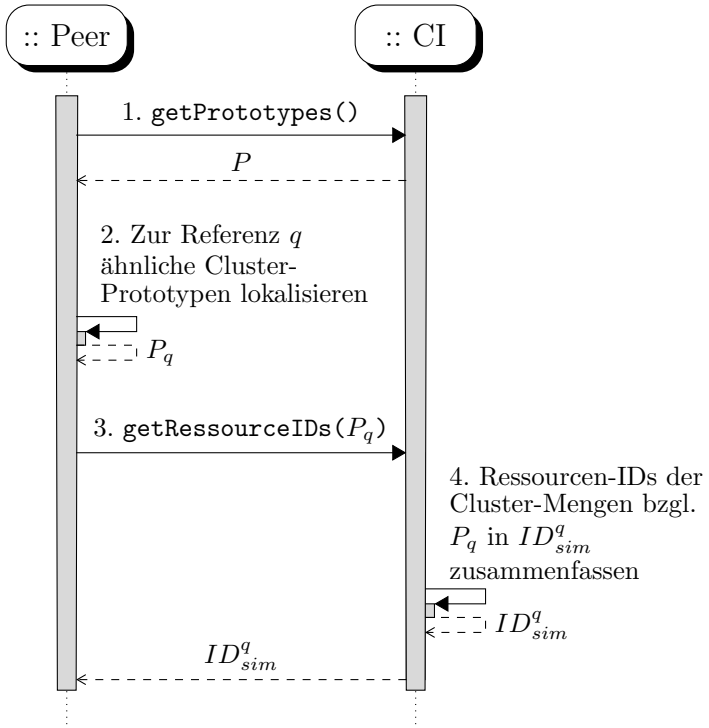


Abbildung 4.2.10: Ablauf der `locate()`-Operation im CLDM bezüglich einer Referenzspezifikation q (Variante B)

das Entfernen von Ressourcen aus Clustern ist es sehr wahrscheinlich, dass sich der Clusterschwerpunkt verschiebt und der betroffene Prototyp angepasst werden muss. (Siehe hierzu Abbildung 4.2.12.) Zu diesem Zweck teilt der Teilnehmer der Cluster-Instanz mit, welche Ressource von ihm entfernt wurde und weist diese an, das betreffende Datum aus ihrer Verwaltung zu entfernen. Die CI muss draufhin erneut ein Clustering durchführen, um gegebenenfalls Änderungen des betroffenen Prototypen Rechnung zu tragen.

Durch diesen Ansatz wird die Identifikation ähnlicher Daten vom eigentlichen P2P-Netz entkoppelt und durch eine gesonderte Instanz durchgeführt. Es ist daher sinnvoll weiter Möglichkeiten zu untersuchen, wie eine stärkere Integration der Lokalisation in eine dezentrale Umgebung zu erreichen ist. Da in Peer-to-Peer-Systemen Ressourcen eindeutige IDs besitzen, die mittels Hash-Funktionen errechnet werden, soll im folgenden Abschnitt eruiert werden, ob es möglich ist, auf deren Basis eine ähnlichkeitsbasierte Lokalisation durchzuführen.

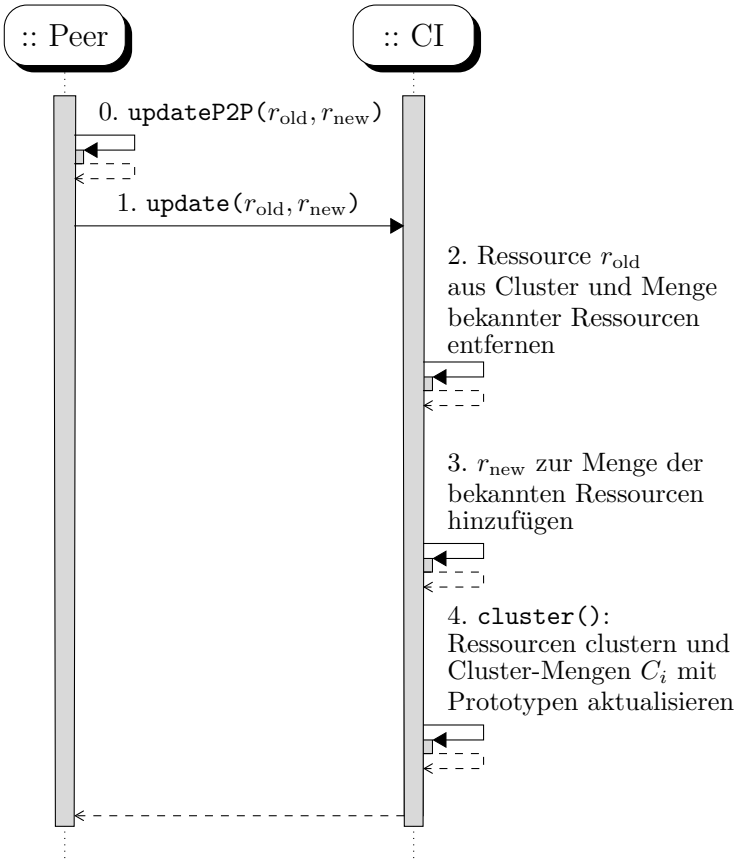


Abbildung 4.2.11: Ablauf der `update()`-Operation im CLDM bezüglich einer alten (r_{old}) und neuen Ressource (r_{new})

4.2.5 Betrachtung existierender P2P-Funktionalität zur Suche ähnlicher Ressourcen

Bei der Ressourcenverwaltung in P2P-Systemen werden Ressourcen ausschließlich über ihren Hash-Wert $h(r)$ identifiziert. Betrachtet man nun das aus Sicht des Anwenders nicht untypische Szenario, Daten in einem solchen System auffinden zu wollen, so kann man die folgenden zwei Fälle bezüglich der Eigenschaften der zu suchenden Ressource r unterscheiden.

1. **Eigenschaften von r sind genau bekannt** Der suchende Peer kann durch Bestimmung von $h(r)$ die gesuchte Ressource im Netzwerk eindeutig ausfindig machen und sie, sofern diese existiert, anfordern. Dieser Vorgang ist dabei in seinem Ablauf weitestgehend trivial.

2. **Eigenschaften von r sind *nicht* genau bekannt** Angenommen der Peer sucht alle Ressourcen r_{sim} die innerhalb einer gewissen Toleranz

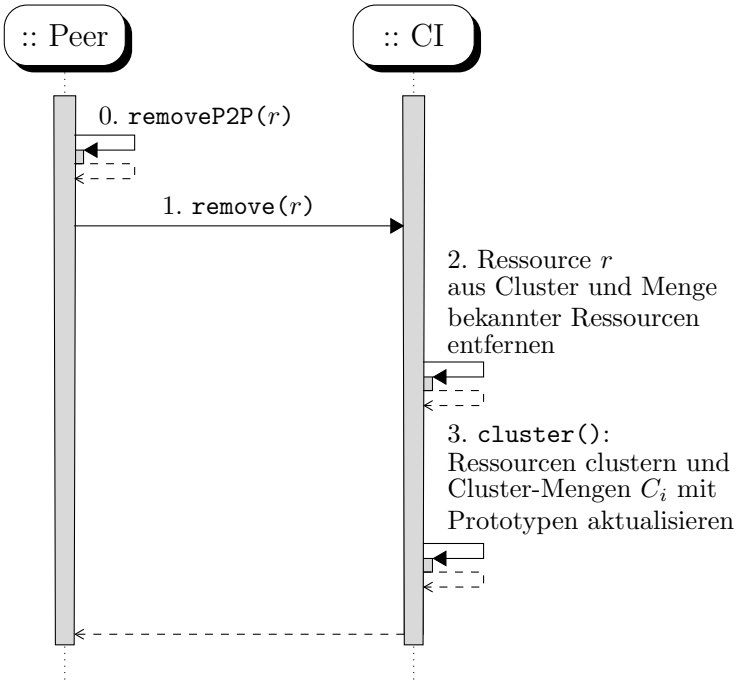


Abbildung 4.2.12: Ablauf der `remove()`-Operation im CLDM bezüglich einer Ressource r

$\varepsilon > 0$ ähnlich zu r sind. Da r_{sim} nicht bekannt ist, müsste der Peer alle Ressourcen-IDs suchen, für die gilt, dass

$$|h^{-1}(ID^r_{\text{sim}}) - r| \leq \varepsilon$$

ist. Er bestimmt also die Umkehrfunktion bezüglich der in dem P2P-Netzwerk verwendeten Hash-Funktion und berechnet von allen (vorhandenen) IDs die ursprünglichen Ressourcen. Anschließend kann er die Ressourcen auswählen, deren Abweichung zu seiner Referenzspezifikation innerhalb der gewünschten Toleranz liegt.

Bei der Umsetzung von Punkt 2 stößt der Peer allerdings auf ein **nichttriviales Problem**: Aus Bedingung (2.4.4) folgt, dass Hash-Funktionen nicht trivial umkehrbar sind, sprich deren Umkehrfunktion nicht berechenbar ist. Eine weitere Möglichkeit, die erschöpfende Suche nach allen infrage kommenden Schlüsseln, die einen geeigneten Hashwert liefern, ist jedoch gerade bei Ausgabelängen gebräuchlicher Hash-Funktionen (128 bis 512 Bits) und dem exponentiellen Suchaufwand praktisch nicht durchführbar. Gerade weil Hash-Funktionen Kollisionen vermeiden und Sicherheit gegen Umkehrbarkeit bieten sollen, scheiden solche Ansätze aus.

Da also Ansätze, um Daten in strukturierten P2P-Netzwerken bezüglich ihrer Ähnlichkeit zueinander finden zu können, mit den hierbei verwendeten,

gleichverteilten Hash-Funktionen nicht zielführend sind, sollen im folgenden Abschnitt spezielle Hash-Funktionen im Hinblick auf die Lösbarkeit dieses Problems untersucht werden.

4.2.6 Locality-Sensitive Hash-Funktionen

In diesem Abschnitt soll untersucht werden, inwiefern das in Abschnitt 2.4.2 vorgestellte Verfahren der approximativen LSH-Heuristik geeignet ist, die im vorherigen Abschnitt beschriebene Problematik der Ähnlichkeitssuche in strukturierten P2P-Systemen zu lösen.

Die LSH-Heuristik dient in erster Linie dazu, ähnliche Daten schnell lokalisieren zu können. Diese Eigenschaften können nach [KIW04, S. 116] genutzt werden, um beispielsweise in nahezu linearer Zeit AkNN-Graphen zu erzeugen. Hierzu werden vorab, wie in Abschnitt 2.4.2 dargestellt, alle Datenpunkte mit den zufällig defnierten Hash-Funktionen $g \in \mathcal{H}$ gehasht und entprechenden Buckets zugewiesen. Anschließend können für jeden Datenpunkt seine k nächsten Nachbarn in $\mathcal{O}(nl|B|) < \mathcal{O}(n^2)$ durch Kanten verbunden werden. Ein solcher approximativer 3-NN-Graph ist in Abbildung 4.2.13 dargestellt. Hierbei wurde das LSH-Verfahren mit den Parametern $l = 5$ sowie $\widetilde{k} = 5$ auf die abgebildete Datenmenge angewendet. Es ist ersichtlich, dass im Unterschied zum exakten 3-NN-Graphen von dem approximativen Verfahren lediglich eine Kante fehlerhaft gezogen wurde (rot eingefärbt) und somit in diesem Beispiel nur ein relativ geringer Fehler aufgetreten ist.

Die Eigenschaft des LSH sehr schnell Datenpunkte zu lokalisieren, die sich in der *Nähe* zu einem bestimmten Punkt $x \in X$ befinden, lässt sich auch nutzen um Kandidaten $s \in X_x^{\text{cand}}$ zu finden, welche **ähnlich** zu einer Suchanfrage x sind:

$$s \in X_x^{\text{cand}} \Leftrightarrow \exists g \in \mathcal{H} : g(s) = g(x)$$

Das bedeutet, dass alle Datenpunkte, für die mindestens eine Hash-Funktion aus $\mathcal{H}$ zu einer Kollision mit x führt, Kandidaten sind.

Ist die Menge X über einem metrischen Vektorraum definiert und sind somit Metriken definierbar (siehe Abschnitt 4.2.1), identifiziert LSH alle Punkte s als *ähnlich* zu einer Suchanfrage x, für die

$$d^{LSH}(x, s) \leq \varepsilon$$

gilt. Dabei lässt sich ε als ein Kollisionsradius um x verstehen, in dem alle Punkte s enthalten sind.

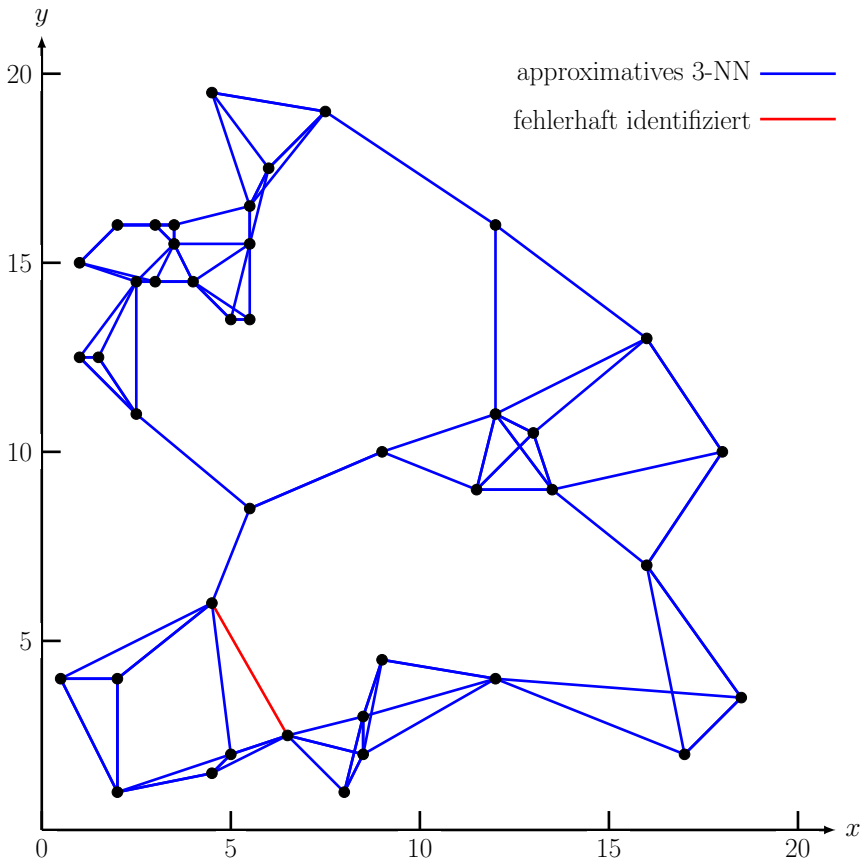


Abbildung 4.2.13: Darstellung eines approximativen LSH-3-NN Graphen

Aus dieser Menge $X_x^{\text{cand}} \ll |X|$, die signifikant kleiner ist als die Ausgangsmenge X, können die für das Problem (beispielsweise kNN oder Ähnlichkeitssuche) relevanten Datenpunkte mit exakten Verfahren, wie dem *euklidischen Abstand* bestimmt werden. Durch die starke Einschränkung des zu betrachtenden Suchraumes, können in diesem exakte Abstandsmessungen (siehe auch Definition 2.3.4) durchgeführt werden, ohne dass eine drastische Verschlechterung der Laufzeit zu erwarten wäre.

4.2.7 Locality-Sensitive Data Model

Auf der Grundlage des LSH soll ein *Locality-Sensitive Data Model (LSDM)* konstruiert werden, um das, in Abschnitt 4.1.1 beschriebene, Problem der Lokalisierung approximativ lösen zu können. Das Ziel soll dabei sein, auf eine zentrale Cluster-Instanz verzichten zu können, um einen „*Flaschenhals*“ bzw. *Single Point of Failure* zu vermeiden. Zudem bietet es sich für eine effiziente Datenverwaltung an, den Geschwindigkeitsvorteil des LSH zu nutzen.

Für das Design eines solchen Datenmodells ist es zunächst notwendig Anforderungen festzulegen, die das Modell zu erfüllen hat.

Anforderungen an das LSDM Bezüglich des zu entwerfenden Datenmodells folgende Anforderungen wichtig:

- Werden Ähnlichkeitsinformationen dem LSDM übergeben, so muss es diese in jedem Fall annehmen und verarbeiten.
- Ähnliche Ressourcen müssen zu einer Referenzspezifikation gefunden werden können.
- Die Modifikation von Ressourcen ist zu unterstützen.
- Wurden Ressourcen aus dem P2P-Netz entfernt, so sind diese ebenfalls aus dem LSDM zu entfernen.

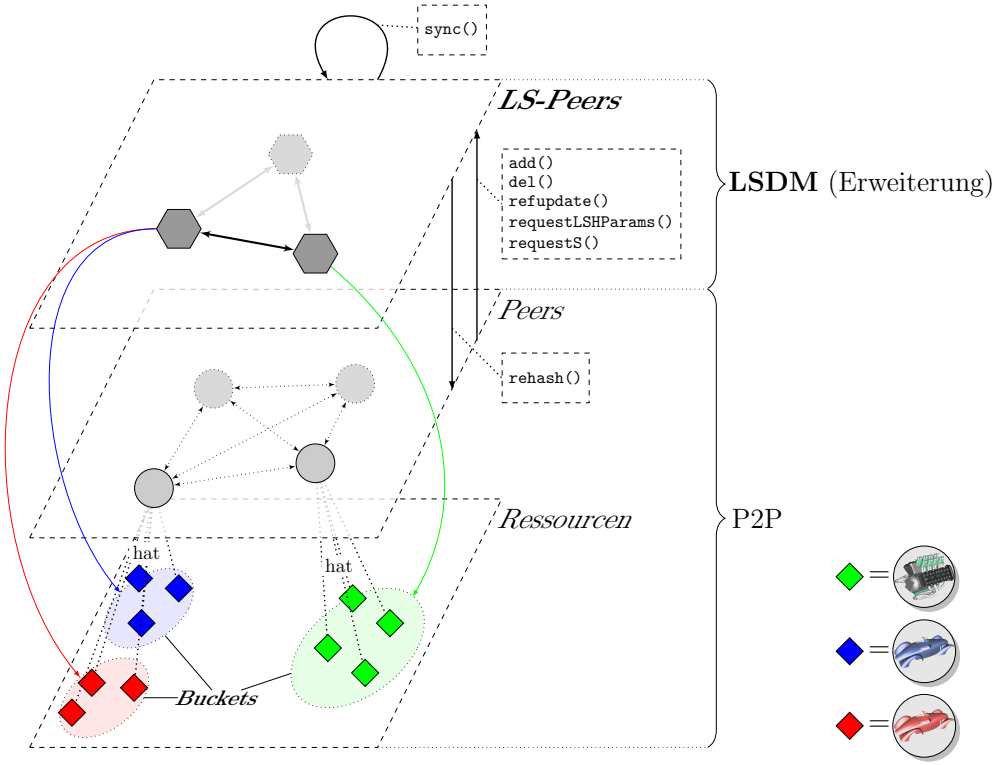


Abbildung 4.2.14: Die 3-Schichten-Architektur des LSDM (Baugruppen aus [Sie12b])

Die konkrete Erweiterung besteht in der Verwaltung von **Ähnlichkeitslisten** durch Peers. Es existieren durch die Modifikation also Peers, die in dem Netzwerk eine gesonderte Stellung einnehmen, da sie neben der Verwaltung allgemeiner Ressourcen auch für die Ähnlichkeitslisten zuständig sind. Daher kann hierbei von einem **Super-Peer**-Netzwerk gesprochen werden. Ein solcher Super-Peer wird, zur besseren Abgrenzung gegenüber einem gewöhnlichen Teilnehmer des Netzwerkes (Peer), im weiteren Verlauf des Verfahrens als Locality-Sensitive Peer (LS-Peer) bezeichnet.

Das LSDM soll, wie in Abbildung 4.2.14 dargestellt, als eine 3-Schichten-Architektur, bestehend aus *Ressourcen-*, *Peer-*, sowie *LS-Peer*-Schicht, modelliert werden. Hierbei gelten die gleichen Annahmen über das zugrundeliegende P2P-System, wie sie bereits für das CLDM (Abschnitt 4.2.4) festgelegt wurden.

Zur Realisierung des LSDMs soll das, in Abschnitt 2.5.5 vorgestellte, Verfahren zur approximativen kNN-Bestimmung so erweitert werden, dass es auf ein bestehendes P2P-System aufsetzen kann. Das Ziel ist, wie auch bei dem Entwurf des CLDMs (Abschnitt 4.2.4), eine Austauschbarkeit des dezentralen Backends sowie eine Reduzierung der Komplexität und Fokussierung auf die relevanten Funktionen zu erreichen. Die Erweiterung des LSH-Verfahrens soll bei der Erstellung der einzelnen Buckets, wie in Abbildung A.2 (Anhang) dargestellt, ansetzen.

Ein Bucket wird dabei eindeutig durch einen Hashwert h aus $\{0,1\}^{\widetilde{k}}$ identifiziert und verwaltet somit alle paarweisen Kollisionen (d.h. potentielle Ähnlichkeiten) zweier Datenpunkte bezüglich eines Hash-Wertes. Es stellt somit die kleinste Zuordnungseinheit dar, Verknüpfungen zwischen einem Referenzpunkt q (Suchanfrage) und den zu diesem ähnlichen Punkte abzubilden. Um in einem strukturierten P2P-Netz die Verwaltungslast solcher Buckets, also der Information welcher Datenpunkt x_i ähnlich zu x_j ist, zu vermindern, sollen diese auf möglichst viele Peers verteilt werden. Hierbei wird deutlich, dass das LSH-Datenmodell auf der bereits bestehenden Funktionalität eines P2P-Netzwerkes aufsetzen und eine Erweiterung dessen darstellen soll. Die Verteilung der einzelnen Buckets auf Peers wird dabei erreicht, indem der Bucket-Identifikator h_i wie eine beliebige Ressource dem zugrunde liegenden P2P-System zur Verwaltung übergeben wird. Es wird somit überprüft, welcher Peer mit der ID $f_X(p_j)$ für den Hashwert der Bucket-ID $f_X(h_i)$ verantwortlich ist. Dies kann durch Fingertabellen (wie in Chord) oder auch der minimalen Hamming-Distanz (siehe Gleichung (2.3.15)) zwischen Ressourcen- und Peer-ID (siehe [MM02]) bestimmt werden.

Eine solche Zuteilung der Verantwortlichkeiten ist in der Abbildung A.2 vereinfacht als Funktion

$$f_{\mathcal{P}}(f_X(h_i)) \rightarrow f_X(p_j)$$

dargestellt. Wurde eine geeignete Zuordnung gefunden, so wird von dem jeweiligen Peer eine Liste S_{h_i} bestehend aus den Ressourcen-IDs des Buckets h_i, also den gehashten Datenpunkten, verwaltet.

Sollen nun ähnliche Daten zu einem gegebenen Datum q lokalisiert werden, so muss bestimmt werden, welchen Buckets q zugeordnet werden muss und

das Netzwerk nach den entsprechenden Ressourcen mit den IDs, die sich aus der Berechnung von $f_X(ID_{h_i^q})$ ergeben, befragt werden. Alle hierfür verantwortlichen LS-Peers teilen daraufhin dem fragenden Peer die Ressourcen-IDs ihrer Buckets mit, die zu der Anfrage h_i^q passen. Der suchende Peer muss jetzt lediglich, wie in dem LSH-Verfahren vorgesehen, diese Mengen $S_{h_i^q}$ vereinigen und erhält auf diese Weise effizient alle zu q ähnlichen Ressourcen.

Um die eingangs aufgestellten Anforderungen erfüllen zu können, soll hier näher spezifiziert werden, wie ein bestehendes strukturiertes P2P-System erweitert werden muss, um die ähnlichkeitsbasierte Verwaltung von Datenobjekten ermöglichen zu können. Der Umfang bzw. die Anzahl der in der Konsequenz zu modifizierenden Komponenten hängt dabei von dem zugrundeliegenden System ab und kann, je nach dessen Komplexität, variieren.

Mit Hilfe der folgenden Operationen soll die Funktionsweise des LSDMs beschrieben werden. Die genaue Reihenfolge und die Abläufe der Kommunikation zwischen Peers und LS-Peers sind zudem in den Abbildungen 4.2.16 bis 4.2.18 sowie Abbildung A.5 in Form von Sequenzdiagrammen visualisiert. Zur vollständigen Einordnung des LSDMs sei auf die Abbildung 4.2.15 verwiesen. Im Anhang findet sich diese Abbildung zudem in vergrößerter Darstellung (Abbildung A.3). Im Weiteren sollen die Ressourcen, bzw Datenobjekte, die

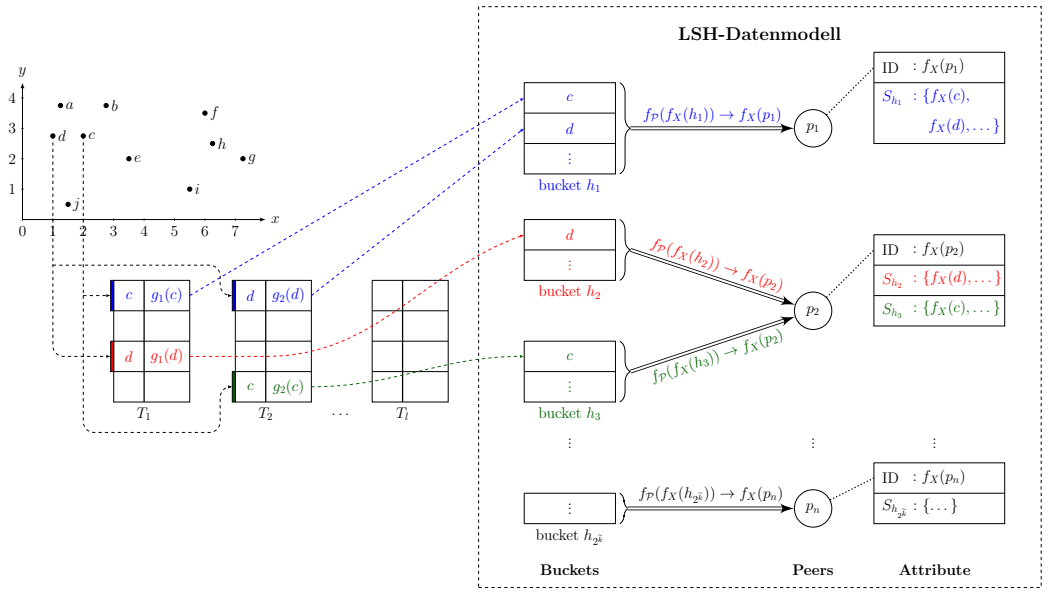


Abbildung 4.2.15: LSH-basiertes Datenmodell

für die LSDM-Operationen notwendig sind, gemäß der Tabelle 4.2.3 benannt werden.

Die Operationen für die Umsetzung des LSDMs sollen dabei im Folgenden spezifiziert werden (Tabelle 4.2.4) und durch entsprechende Sequenzdiagramme näher erläutert werden.

Ressourcen (-mengen)	
r	allgemeine Bezeichnung einer Ressource
r_{new}	Ergebnis einer Ressourcenmanipulation
r_{old}	Ausgangspunkt einer Ressourcenmanipulation
q	Referenzspezifikation einer Suchanfrage (*Query*)
r_{cand}	Kandidat für eine ähnliche Ressource bzgl. q
R_{q}	Menge ähnlicher Ressourcen zum Query q

Hash-Werte und -Funktionen	
h	Hashwert einer LSH-Funktion
g	LSH-Funktion $g \in \mathcal{H}$
f_X	gleichverteilte Hash-Funktion eines P2P-Systems
$\mathcal{H}$	Menge von LSH-Funktionen $\{g_1, \ldots, g_l\}$

Mengen von Hashwerten	
H	allgemeine Menge von Hashwerten $\{h_1, h_2, \ldots\}$
H_{old}	Hashwerte, die aktualisiert werden sollen
H_{new}	Hashwerte, die aktualisiert wurden

Tabelle 4.2.3: Relevante Bezeichnungen für das LSDM

- **Übergabe von Ressourcen** Eine Übergabe der Ressourcen an das P2P-Netzwerk bedeutet, dass in dem LSDM entsprechende Ähnlichkeitsinformationen hinterlegt werden müssen. Hierbei soll die notwendige Berechnung der LS-Hashwerte von den einzelnen Peers übernommen werden, um die Gesamtlast zu verteilen. Dies bedeutet jedoch, dass initial jeder Peer die zu verwendenden Hash-Funktionen mitgeteilt bekommen muss. Somit muss ein Peer, nachdem er dem Netzwerk beigetreten ist und Ressourcen im Netzwerk bereitstellen will, von einem der vorhandenen LS-Peers die zu verwendenden Hash-Funktionen g anfordern (Abbildung 4.2.16). Aus **Sicht des Peers** gestaltet sich der Ablauf für die Veröffentlichung einer Ressource r dabei wie folgt (Schritt 0 bis 3):

  0. Veröffentliche die Ressource r im Netzwerk, entsprechend der bereits existierenden Funktionalität des vorliegenden P2P-Systems.
  1. Beziehe, sofern diese nicht vorhanden oder aktuell sind, von einem beliebigen LS-Peer mittels `requestLSHParams()` die für Schritt 2 benötigten LSH-Funktionen.
  2. Bestimme die von r zu belegenden Buckets, indem die LSH-Werte mit den Funktionen aus Schritt 1 berechnet werden. Jeder dieser LSH-Werte h identifiziert eine Ähnlichkeitsliste, in die die Ressourcen-ID von r aufgenommen werden muss.
  3. Ermittle die für die jeweiligen Ähnlichkeitslisten verantwortlichen LS-Peers durch die Berechnung von $\text{ID}_h = f_X(h)$. Übertrage an-

LSDM Operationen (Peer→LS-Peer)	
`add`($\mathrm{ID}_h, h, \mathrm{ID}_r$)	Übergib dem LS-Peer mit der ID_h die Ressourcen ID_r, damit dieser sie in Bucket h ablegt.
`remove`($\mathrm{ID}_h, h, \mathrm{ID}_r$)	Weise den LS-Peer ID_h an, die Ressourcen-ID_r aus dem Bucket h zu entfernen.
`requestLSHParams`($\hat{r}$)	Frage von einem LS-Peer die LSH-Funktionen $\mathcal{H}$ an. Soll eine Ressource übergeben werden, die das maximal zulässige Vektorattribut überschreitet, so wird das benötigte Maximum in Form von $\hat{r}$ angegeben.
`requestS`(ID_h)	Fordere die Ähnlichkeitsliste S_h bezüglich der ID_h an.
`refupdate`(ID_h, h, $\mathrm{ID}_{r_{\mathrm{old}}}, \mathrm{ID}_{r_{\mathrm{new}}}$)	Ersetze in Bucket h des LS-Peers ID_h die $\mathrm{ID}_{r_{\mathrm{old}}}$ durch $\mathrm{ID}_{r_{\mathrm{new}}}$.

LSDM Operationen (LS-Peer→Peer)	
`rehash`($\mathcal{H}$,k,l)	Weise einen Peer an, seine Ressourcen mit neuen LSH-Funktionen zu hashen und diese anschließend mittels `refupdate()` zu aktualisieren.

LSDM Operationen (LS-Peer→LS-Peer)	
`initLSH`($\hat{r}$)	Initialisiere neue LSH-Funktionen bezüglich $\hat{r}$.
`sync`(d,$\mathcal{H}$,$\hat{r}$,k,l)	Synchronisiere die erzeugten LSH-Funktionen $\mathcal{H}$ und LSH-Parameter $\hat{r}$,k sowie l mit einem LS-Peer.

P2P-Backend Operationen	
`addP2P`(r)	Füge Ressource r dem Netzwerk hinzu.
`updateP2P`($r_{\mathrm{old}}, r_{\mathrm{new}}$)	Ersetze Ressource r_{old} durch r_{new}.
`removeP2P`(r)	Entferne Ressource r aus dem Netzwerk.

Tabelle 4.2.4: Vorgesehene Operationen für das Locality-Sensitive Data Model

schließend den Hash-Wert ID_r der Ressource r, durch den Aufruf der Methode `add`($\mathrm{ID}_h, h,$ `ID`$_r$), an die verantwortlichen LS-Peers.

Die für die entsprechenden Ähnlichkeitslisten verantwortlichen **LS-Peers** nehmen diese Anfragen entgegen und verarbeiten diese entsprechend wie in Schritt 4 angegeben:

4. Füge $(\mathrm{ID}_p, \mathrm{ID}_r)$ in die Ähnlichkeitsliste S_h ein, die für den LSH-Wert h verantwortlich sind. Hierbei merkt sich der LS-Peer welcher Peer diese Ressource hinzugefügt hat. Dies wird sich bei der Beschreibung der Ausnahmebehandlung als hilfreich erweisen.

- **Lokalisation anhand von Referenzspezifikationen** Ein mit dem Netzwerk verbundener Peer soll in der Lage sein, zu einem ihm bekannten Referenzspezifikation q, die **IDs aller ähnlichen Ressourcen** zu erhalten, die im Netzwerk verfügbar sind. Hierzu geht er wie in Abbildung 4.2.17 (Schritt 1 – 3) beschrieben vor.

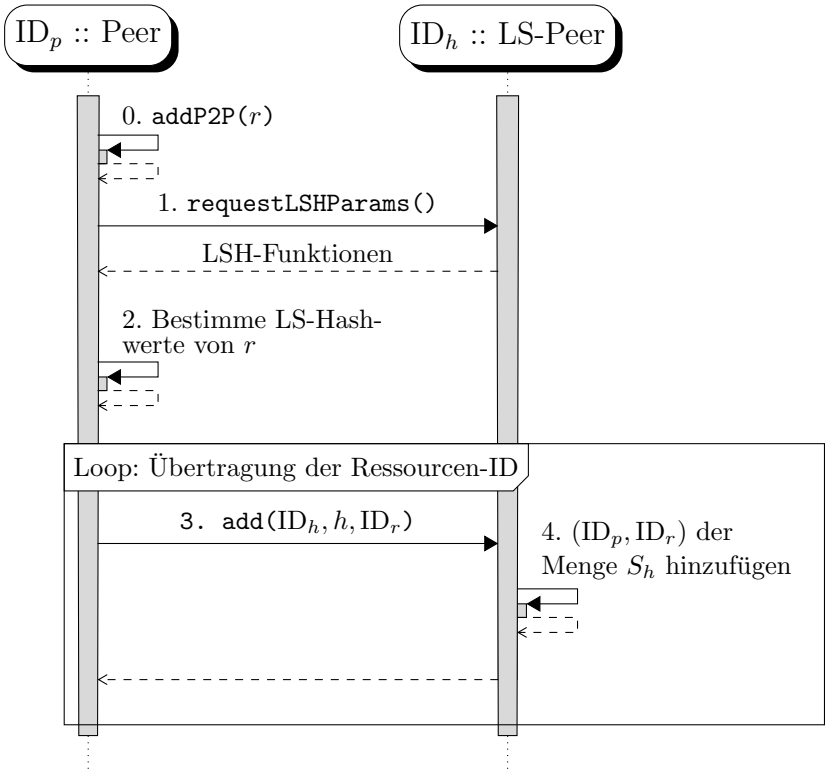


Abbildung 4.2.16: Ablauf der `add()`-Operation im LSDM bezüglich einer Ressource

1. Beziehe, sofern diese nicht vorhanden oder aktuell sind, von einem beliebigen LS-Peer mittels `requestLSHParams()` die für Schritt 2 benötigten LSH-Funktionen.
2. Bestimme ausgehend von q die Liste aller LSH-Werte h, welche die Buckets identifizieren, in denen sich zu q ähnliche Ressourcen befinden. Die für die einzelnen Bucket-IDs h verantwortlichen LS-Peers lassen sich ermitteln, indem q mit denen in Schritt 1 erhaltenen LSH-Funktionen gehasht wird.
3. Fordere, ausgehend von den verantwortlichen LS-Peers die Ähnlichkeitslisten S_h an und füge sie in die Menge der IDs der zu q ähnlichen Ressourcen S ein.
4. Frage das P2P-Netzwerk nach den Ressourcen, die über die IDs aus S identifiziert werden und erhalte somit alle zu q ähnlichen Ressourcen.

- **Modifikation von Ressourcen** Ändern sich Ressourcen in einem P2P-Netzwerk, so hat dies auch Auswirkungen auf die Ähnlichkeit zu anderen Datenobjekten. Aus diesem Grund müssen die Zuordnungen der ID von r_{old} zu den Ähnlichkeitslisten S_{h_i} überprüft und ggf. aktualisiert

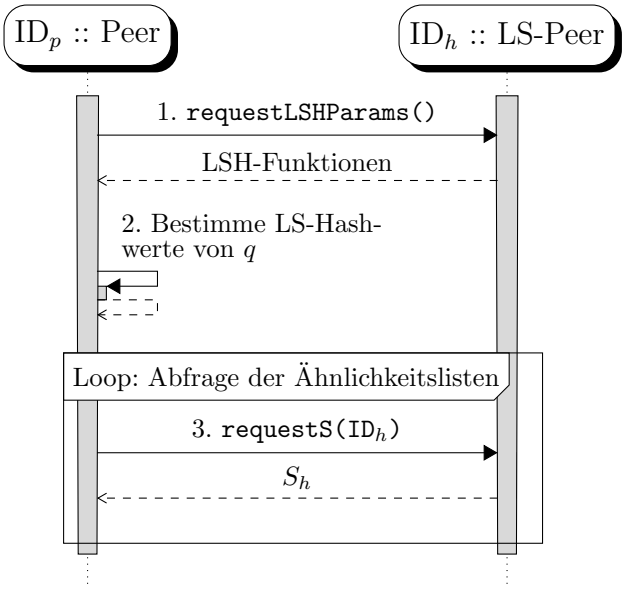


Abbildung 4.2.17: Ablauf der Lokalisations-Operation im LSDM bezüglich einer Ressource

werden. Dabei müssen in **allen** Ähnlichkeitslisten S_h vorhandene Tupel $(\mathrm{ID}_p, \mathrm{ID}_{r_{\mathrm{old}}})$ durch $(\mathrm{ID}_p, f_X(r_{\mathrm{new}}))$ ersetzt werden bzw. abhängig vom Grad der Unterschiede zwischen alter und neuer Ressource entfernt oder neu eingefügt werden.

Bei der Verwaltung von Datenobjekten durch das LSDM kann es von Vorteil sein, diese *in place* (also direkt) modifizieren zu können, um eine Verkettung von zwei Operationen (`remove()` und `add()`) zu vermeiden. Jede Liste S_{h_i} muss somit nur einmal betrachtet werden, wodurch vermeidbare Anfragen im Netzwerk eingespart werden können. Der Ablauf soll dabei (siehe Abbildung A.4) wie folgt aussehen:

0. Ist $r_{\mathrm{old}} = r_{\mathrm{new}}$, so breche ab.
1. Beziehe, sofern diese nicht vorhanden oder aktuell sind, von einem beliebigen LS-Peer mittels `requestLSHParams()` die für Schritt 2 benötigten LSH-Funktionen.
2. Bestimme die zu aktualisierenden Buckets, durch Berechnung aller LS-Hashwerte h_{old} für die alte und h_{new} für die neue Ressource. Platziere dabei alle Werte für h_{old} in der Menge H_{old}, sowie analog die Werte h_{new} in H_{new}.
3. Es kann in drei Arten von Anfragen unterschieden werden, die an die für die Ressourcen verantwortlichen LS-Peers geschickt werden müssen, um deren Ähnlichkeitslisten zu aktualisieren:

a) **Neu einzutragende Ähnlichkeiten** Kommen zu denen in H_{old} neue LSH-Werte hinzu, so müssen diese mit Hilfe der von den LS-Peers bereitgestellten Methode `add`$(\text{ID}_h, \text{ID}_{r_{\text{new}}}, h)$ übermittelt werden. Dabei sind als Parameter die ID für den verantwortlichen LS-Peer (ID_h), der Bezeichner h für die Ähnlichkeitsliste S_h, sowie die ID der Ressource r_{new} anzugeben.

b) **Zu entfernende Ähnlichkeiten** LSH-Werte, die aus bestehenden Ähnlichkeitslisten entfernt werden müssen, zeichnen sich dadurch aus, dass sie zwar in der alten LS-Hashmenge H_{old} enthalten sind, jedoch **nicht** mehr in der neuen H_{new}. Es müssen also alle LS-Peers, die für diese – jetzt überflüssigen – Werte zuständig sind, aufgefordert werden, die Referenzen auf nun zu r unähnlichen Ressourcen aus ihren Listen zu entfernen. Dies geschieht, indem ihnen durch den Aufruf der Methode `del`$(\text{ID}_h, h, \text{ID}_{r_{\text{new}}})$ (analog zum Aufruf von `add()`) die Entfernung der entsprechenden Referenzen mitgeteilt wird.

c) **Zu ersetzende Ähnlichkeiten** Wurden an der Ressource r_{old} nur geringfügige Änderungen vorgenommen, so kann davon ausgegangen werden, dass der überwiegende Anteil an LS-Hashwerten gleich bleibt, d.h. viele Referenzen auf r_{new} in den selben Buckets und somit auch Ähnlichkeitslisten bei den LS-Peers, verbleiben. Für diesen Fall müssen den betroffenen LS-Peers die für die Schnittmenge $H_{\text{old}} \cap H_{\text{new}}$ verantwortlich sind, die Änderung der Referenz auf die neue Ressource r_{new} mitgeteilt werden. Es muss also die Methode `refupdate`$(\text{ID}_h, h, \text{ID}_{r_{\text{old}}}, \text{ID}_{r_{\text{new}}})$ ausgeführt werden, um die LS-Peers zu veranlassen den Eintrag in Liste S_h von $\text{ID}_{r_{\text{old}}}$ auf $\text{ID}_{r_{\text{new}}}$ zu ändern.

- **Entfernen von Ressourcen** Sollen Ressourcen aus dem P2P-System entfernt werden, so müssen sich diese Änderungen natürlich auch im LSDM widerspiegeln. Hierzu müssen Referenzen auf gelöschte Ressourcen auch aus den Ähnlichkeitslisten der LS-Peers entfernt werden. Dies soll durch folgenden Ablauf garantiert werden, den ein Peer anstoßen muss, wenn er eine Ressource aus dem Netzwerk entfernt (Abbildung 4.2.18):

  0. Entferne Ressource r aus dem P2P-System gemäß der hierfür spezifizierten Vorgehensweise.

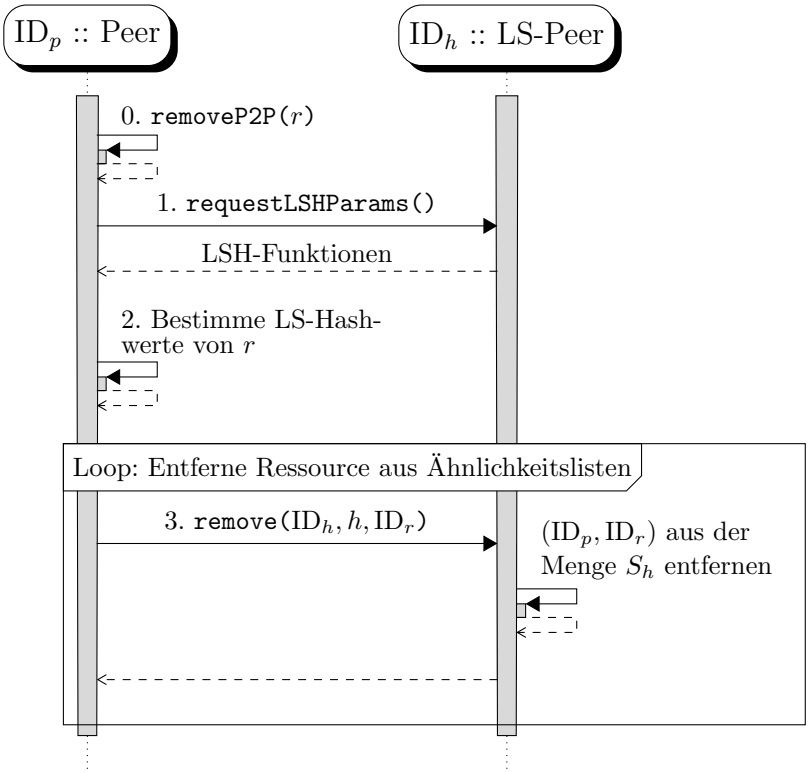


Abbildung 4.2.18: Ablauf der `remove()`-Operation im LSDM

1. Beziehe, sofern diese nicht vorhanden oder aktuell sind, von einem beliebigen LS-Peer mittels `requestLSHParams()` die für Schritt 2 benötigten LSH-Funktionen.
2. Bestimme die von r belegten Buckets, indem die LS-Hashes mit den Funktionen aus Schritt 1 berechnet werden. Jeder der LSH-Werte h bezeichnet (analog zu `add()`) eine Ähnlichkeitsliste, aus der ID_r entfernt werden muss.
3. Die für die einzelnen Listen S_h verantwortlichen LS-Peers lassen sich ermitteln, indem r mit denen in Schritt 1 erhaltenen LSH-Funktionen gehasht wird. Durch den Aufruf von `remove`$(\mathrm{ID}_h, \mathrm{ID}_r, h)$ werden anschließend die Referenz-Tupel $(\mathrm{ID}_p, \mathrm{ID}_r)$ aus den Listen S_h vom angefragten LS-Peer entfernt.

- **Ausnahmebehandlung bei Überschreitung des maximal zulässigen Vektorattributes** Will ein Peer eine Ressource r hinzufügen, deren maximaler Attributwert größer als der maximal zulässige ist, so muss ein *Rehashing* durchgeführt werden. Dies liegt in der Tatsache begründet, dass die LS-Hash-Funktionen, wie in Gleichung (2.4.6) dargestellt, auf einer unären Vektordarstellung operieren und somit der erlaubte Wertebereich, durch die zur Verfügung stehende Anzahl an Stelle je unärem

Attribut, limitiert ist. Zum besseren Verständnis sei an dieser Stelle auf die Abbildung 2.4.2 verwiesen, in der eine exemplarische Berechnung eines LS-Hash-Wertes durchgeführt wird.

Tritt nun der Fall ein, dass ein maximal zulässiger Wert $\hat{x}$ durch $\max_i(r_i)$ überschritten wird, so lässt sich r in keine, für die LS-Hash-Funktionen passende, unäre Darstellung überführen. Aus diesem Grund bleiben an dieser Stelle nur drei Möglichkeiten zur Lösung des Problems:

1. **Ablehnung der Ressource von LSDM und P2P-Netz** Da die Hauptaufgabe eines Produktmodells die Verwaltung von Daten ist, ist eine vollständige Ablehnung die am wenigsten praktikable Lösung und sollte – sofern nicht triftige Gründe, wie z.B. unberechtigte Operationen, vorliegen – unbedingt vermieden werden.

2. **Annahme der Ressource vom P2P-Netz, aber Ablehnung durch LSDM** Dies hätte zur Folge, dass die Ressource, wenn direkt nach ihr gesucht wird, zwar noch über das P2P-Netz auffindbar ist, diese jedoch nicht bei Anfragen nach ähnlichen Ressourcen über das LSDM lokalisiert werden kann. Ein solcher Ansatz bewirkt, dass eine Ressourcenannahme, wie in den Anforderungen definiert, nicht ohne Einschränkung des zulässigen Wertebereiches garantiert werden kann. Stünde der Wertebereich von vornherein fest und wäre in einem entsprechenden Anwendungsszenario mit einer Überschreitung nicht zu rechnen, so stellt dies eine denkbare, jedoch eher unflexible Lösung dar.

3. **LSDM und P2P-Netz akzeptieren beide die Ressource** Ist dies zwar das aus Sicht der Teilnehmer wünschenswerteste Ergebnis, so bringt es doch den meisten Verwaltungsaufwand mit sich. Als erstes muss ein neuer Wert für $\hat{x}$ festgelegt werden. Hierfür bietet es sich an, nicht bloß eine minimale Erhöhung auf $\max_i(r_i)$ vorzunehmen, sondern, um allzu häufigen Überschreitungen vorzubeugen, einen gewissen *Puffer* nach oben einzuräumen. Dieser sollte jedoch nicht zu groß gewählt werden, da bei bereits vorhandenen Daten die Anzahl konkatenierter Nullen in der unären Vektordarstellung entsprechend zunimmt. Somit läuft man leicht Gefahr, dass die Kollisionswahrscheinlichkeit zunimmt und hierdurch die Anzahl von *false-positives* steigt.

   Da durch die längere unäre Darstellung die LSH-Funktionen nicht mehr kompatibel bezüglich der einzustellenden sowie existierenden

Ressourcen sind, müssen alle zuvor festgelegten und an LS-Peers replizierten Funktionen $\mathcal{H}$ für ungültig erklärt und neue bestimmt werden. Zu diesem Zweck generiert der LS-Peer, wie in Abbildung A.5 (Anhang) dargestellt, neue Hash-Funktionen und repliziert diese an alle übrigen LS-Peers. Um sicherzustellen, dass alle LS-Peers mit aktuellen $\mathcal{H}$ operieren, versieht der initiierende LS-Peer seine Replikationsnachricht mit einem Zeitstempel. Anschließend müssen alle LS-Peers die für die Ressourcen-IDs in den Ähnlichkeitslisten verantwortlichen Peers auffordern, einen `rehash()` mit den neuen LSH-Funktionen durchzuführen. Ist dies von Peer-Seite abgeschlossen, so führen diese ein `refupdate()` durch, um ggf. Änderungen der Ähnlichkeiten Rechnung zu tragen. Besteht das LSDM nun wieder in einem konsistenten Zustand, so kann von dem eingangs beschriebenen Peer durch ein `add()` die Ressourcen-ID von r an den verantwortlichen LS-Peer übergeben werden.

Ausfall von LS-Peers Gerade in Netzwerken die potentiell starken Fluktuationen bezüglich angemeldeter Nutzer unterworfen sind, ist es sinnvoll sich Gedanken über Strategien im Falle von nicht mehr erreichbaren Teilnehmern zu machen. In dem entworfenen Datenmodell, welches auf einem vorhandenen P2P-System basiert, wurde dies für die benötigten LS-Peers noch nicht getan und soll in diesem Abschnitt erfolgen.

Bisher sieht das LSDM vor, dass jeder Peer anhand des P2P-Hash-Wertes über einer Bucket-ID $f_X(\mathrm{ID}_h)$ den hierfür verantwortlichen LS-Peer findet. Dies geschieht über die gegebene Funktionalität des zugrundeliegenden P2P-Systems, verantwortliche Teilnehmer anhand von Ressourcen-IDs zu lokalisieren. Ist nun ein LS-Peer nicht mehr verfügbar, so sind die von ihm verwalteten *Ähnlichkeitslisten* nicht mehr verfügbar. Dies hat zur Konsequenz, dass die verwalteten Referenzen nicht mehr im Netzwerk vorhanden sind und diese Referenzen – sollten sie infrage kommen – somit bei Suchanfragen nach ähnlichen Ressourcen nicht mehr an den suchenden Peer übermittelt werden können. Der Peer erhält somit unvollständige bzw. schlimmstenfalls keine Suchergebnisse. Für den Ausfall eines LS-Peers lassen sich zwei mögliche Ursachen ausmachen, die einer unterschiedlichen Behandlung bedürfen:

1. **Planmäßige Abmeldung** Meldet sich ein LS-Peer gemäß des vom P2P-System vorgegebenen Protokolls ordnungsgemäß ab, so muss sichergestellt werden, dass die von ihm verwalteten Ähnlichkeitslisten nicht verloren gehen. D.h. es muss ein „Nachfolger“ gefunden werden, der verant-

wortlich für die IDs der Ähnlichkeitslisten des scheidenden Peers ist. Je nach verwendetem P2P-System kann dieses auf unterschiedliche Arten geschehen – so kann zum Beispiel bei Netzwerken, die in Ringstrukturen organisiert sind, der direkte Nachfolger gewält werden.

2. **Unplanmäßiger Ausfall** Der spontane Ausfall eines Peers stellt ein nicht zu vernachlässigendes Szenario dar. Dies macht sich in der Regel dadurch bemerkbar, dass dieser entweder nicht mehr erreichbar ist, oder auch unsinnige Antworten liefert und in beiden Fällen unbrauchbar ist. Tritt dieser Fall ein, so können Ähnlichkeitslisten **nicht** vorzeitig auf andere LS-Peers gesichert werden. Es ist somit also sicherzustellen, dass für eine solche Situation vorgesorgt ist, indem Sicherheitskopien der Ähnlichkeitslisten auf weitere LS-Peers verteilt werden. Eine denkbare Lösung für Netzwerke die in Ringstrukturen organisiert sind, wäre es, die Nachbarn, die automatisch im Falle eines Ausfalls für die zu sichernden IDs der Listen verantwortlich wären, als Backup-LS-Peers zu bestimmen. In Netzwerken, in denen die Verantwortlichkeit für Ressourcen anhand der numerischen Ähnlichkeit zwischen Peer-ID und Ressourcen-ID festgelegt wird, ließe sich dies implementieren, indem beispielsweise die n numerisch nächsten Teilnehmer gewählt werden.

4.3 Evaluation

Die in Abschnitt 4.2.3 vorgestellten Clustering-Verfahren sowie die in Abschnitt 4.2.4 und 4.2.7 entworfenen Datenmodelle (CLDM und LSDM) zur ähnlichkeitsbasierten Lokalisation von Ressourcen in P2P-Netzwerken werden in den folgenden Abschnitten evaluiert.

4.3.1 Ausgewählte Clustering-Verfahren

In diesem Abschnitt sollen zunächst die Anforderungen dargelegt werden, die für die Untersuchung der Verfahren relevant sind. Anschließend erfolgt eine Bewertung der Clustering-Algorithmen, in der erläutert wird, welche Verfahren für den Einsatz im Rahmen der CLDM geeignet sind.

Anforderungen Das Ziel der zu untersuchenden Algorithmen soll sein, neben der Gruppierung von Datenobjekten und Netzgraphen, für jede Gruppe von Daten ein prototypisches Exemplar (siehe Abschnitt 4.2.2) beziehungsweise im Falle von Netzgraphen einen Repräsentanten (siehe Abbildung 5.1.3, Abschnitt 5.1) zu bestimmen.

Verfahren, die weder Prototypen noch Repräsentanten bestimmen können, werden ausgeschlossen und nicht näher betrachtet. Der Grund liegt darin, dass bei solchen Verfahren ein zusätzlicher Aufwand zur nachträglichen Bestimmung von Prototypen bzw. Repräsentanten erforderlich wäre. Dieser kann jedoch kaum gerechtfertigt werden, wenn alternative Verfahren existieren, die die obigen Anforderungen erfüllen.

Ant Colony Optimization Die ACO stellt ein interessantes Verfahren dar, da sie dezentral implementiert werden kann. Somit kann sie gerade in verteilten Systemen, wie den eingangs angeführten *kollaborativen Produktentwicklungsumgebungen* (Abschnitt 4.1.2) sinnvoll eingesetzt werden, um die Last eines Clusterings auf mehre Teilnehmer zu verteilen. Allerdings ist die Ant Colony Optimization nicht in der Lage Prototypen zu identifizieren und scheidet daher hinsichtlich dieser Anforderungen aus. Jedoch bietet sie die Möglichkeit Gemeinschaften in ungewichteten Netzgraphen zu identifizieren und wird daher in Abschnitt 5.6.2 näher betrachtet.

LSH-Link Der Vorteil gegenüber der „klassischen“ exakten Suche nach benachbarten Punkten liegt bei diesem Ansatz in einer approximativen Bestimmung auf Basis des LSH-Konzeptes (siehe Abschnitt 2.5.5). LSH-Link besitzt daher eine geringere Laufzeit gegenüber exakten hierarchischen Verfahren. Jedoch kann es wie die ACO keine Prototypen bestimmen und ist von daher nicht weiter relevant.

k-means / Relational Neural Gas Beide Verfahren erzeugen, wie bezüglich k-means in Abschnitt 2.7.1 beschrieben, Prototypen, die in den Schwerpunkten der erkannten Cluster liegen. Wurde eine Datenmenge mit diesen oder einem verwandten Verfahren verarbeitet, so können die dabei bestimmten Prototypen als Identifikationsmerkmal für die einzelnen Cluster dienen. Es lassen sich daher bei der alleinigen Betrachtung von Prototypen bereits gewisse Rückschlüsse bezüglich der Relevanz der in den jeweiligen Clustern vertretenden Datenpunkten treffen. Aus diesem Grund wäre dieses Verfahren für die Verwendung innerhalb der CLDM denkbar.

Affinity Propagation Bei AfP handelt es sich zwar um ein graphenbasiertes Clustering-Verfahren, allerdings können Daten durch den Einsatz geeigneter *unterstützender Verfahren* (siehe Abbildung 4.2.6), wie beispielsweise *Mutual k-Nearest Neighbor*, in einen gewichteten Netzgraphen überführt und von der

AfP verarbeitet werden. Hierbei entsteht für jedes Cluster genau ein sogenannter *Repräsentant*, der als Prototyp im Rahmen der CLDM infrage kommt. An dieser Stelle kann ein Mehraufwand zur Bestimmung der Prototypen gerechtfertigt werden, da ein repräsentativer Knoten einem Prototypen entspricht und eine Abbildung von diesem Knoten auf seinen ursprünglichen Datenpunkt trivial möglich ist. Die AfP findet zudem in Abschnitt 5.6.3 bei der Bestimmung von Repräsentanten Verwendung.

Spectral Clustering Da bei dem Clustering mittels Spectral Clustering keine Prototypen bzw. Repräsentanten erzeugt werden, erfüllt es nicht die notwendigen Anforderungen.

Ergebnis Wie bereits in den Anforderungen beschrieben, sollen die Verfahren selbst Prototypen berechnen können. Daher kommen von denen in dieser Arbeit untersuchten Clustering-Verfahren, als Kandidaten für die Cluster-Instanz des CLDM nur *Affinity Propagation* und *k-means* bzw. *Relational Neural Gas* infrage. Die übrigen Verfahren wurden ausgeschlossen, da sie nicht in der Lage waren direkt oder indirekt Prototypen zu bestimmen. Mit einer indirekten Bestimmung ist dabei gemeint, dass, wie im Falle von Affinity Propagation, das Cluster-Problem zunächst in Form eines Netzgraphens abstrahiert wird. In diesem werden anschließend repräsentative Exemplare bestimmt, die im Kontext der Datenstrukturen Prototypen darstellen.

Die Ergebnisse der Evaluation wurden im Hinblick auf die Verwendung innerhalb des CLDM in der Tabelle 4.3.1 noch einmal gegenübergestellt.

Übersicht über untersuchte Clustering-Verfahren				
Verfahren	**Eingabe**	**Prototypen/ Repräsentanten**	**Lokalität der Berechnung**	**Geeignet für CLDM**
Affinity Propagation	Ähnlichkeiten, Distanzen, gewichtete Netzgraphen	Repräsentanten	lokal, verteilt	ja, aber Transformation der Eingabedaten notwendig
Ant Colony Optimization	ungewichtete Netzgraphen	keine	lokal, verteilt (Agenten)	nein
k-means	Vektoren	Prototypen	lokal	ja
Relational Neural Gas	Distanzen	Prototypen	lokal	ja
Spectral Clustering	gewichtete Netzgraphen	keine	lokal	nein
LSH-Link	Vektoren	keine	lokal	nein

Tabelle 4.3.1: Einordnung untersuchter Clustering-Verfahren

4.3.2 Unterstützende Verfahren

Neben den zuvor evaluierten Clustering-Verfahren, werden in diesem Abschnitt die *unterstützenden Verfahren* näher betrachtet. Das Ziel ist dabei die benötigten Laufzeiten unterschiedlicher Verfahren zur Berechnung von paarweisen Ähnlichkeiten bzw. Distanzen zwischen textuellen Daten zu vergleichen sowie die Überführung von Datenpunkten und ungewichteten Netzgraphen in gewichtete Netzgraphen zu evaluieren.

Ähnlichkeits-/Distanzbestimmung textueller Daten Liegen textuelle Daten vor, so müssen Metriken auf diesen definierbar sein, um Clustering-Verfahren anwenden zu können. Dies bedeutet, dass es notwendig ist, Ähnlichkeiten bzw. Distanzen zwischen allen Datenpaaren zu bestimmen. Zu diesem Zweck wurden in Abschnitt 2.3.5 Algorithmen vorgestellt, mit denen jeweils zwei beliebig lange Texte (*Strings*) bezüglich ihrer Ähnlichkeit verglichen werden können. Im Zuge der Evaluation wurden exemplarisch jeweils 22 verwandte Wikipedia-Artikel zu den Themen *„Religion“* und *„CPU“* zufällig ausgewählt und mittels *String kernel*, *NCD* und *CDM* verarbeitet. Hierbei wurden, aufgrund des zu erwartenden hohen Speicherverbrauchs der *String kernel*-Matrizen, lediglich die *Abstracts* der Artikel betrachtet. Die Berechnungen wurden auf einem Testsystem durchgeführt, welches mit einer Intel Xeon E5410 CPU (2,33 GHz) und einer zum Zeitpunkt dieser Arbeit aktuellen Fedora Linux Distribution ausgestattet war. Die im Rahmen dieser Arbeit erstellten Referenzimplementierungen zu den eingesetzten Algorithmen finden sich auf der beigelegten CD.

Laufzeiten						
String kernel (mit $\lambda = 0{,}5$)					**NCD**	**CDM**
Substringlänge n	**2**	**3**	**4**	**5**	(mit *bzip2*)	
Zeit*	701,996 s	2140,189 s	3011,052 s	3860,146 s	1,683 s	1,672 s
Speicher-verbrauch*	≈ 1,9 GB				≈ 5 MB	
*Mittelwert aus 10 Durchläufen, Gesamtgröße aller Artikel: 192 KB						

Tabelle 4.3.2: Laufzeiten zum Vergleich textueller Daten bezüglich 44 Artikeln

Wie in Tabelle 4.3.2 zu erkennen ist, lag die Laufzeit bei der Bestimmung der Ähnlichkeiten mittels *String kernel* deutlich über der von *NCD* und *CDM*. Zudem wurden bei der *String kernel*-Berechnung deutlich mehr Arbeitsspeicher (ca. 1,9 Gigabyte) als bei der *NCD* und *CDM* (ca. 5 Megabyte) belegt.

Der Grund für die hohe Laufzeit liegt in der Komplexität von $\mathcal{O}(k|s||t|)$, wobei k die Länge des Substrings sowie $|s|$ und $|t|$ die Längen der zu vergleichenden Strings angeben (siehe [Lod+02, S. 425]). Insgesamt entsteht also bei

n Datenobjekten (bzw. hier Artikeln) und der Annahme, dass eine maximale Textlänge von $|w|$ zu erwarten ist, ein Aufwand von $\mathcal{O}(n^2k|w|^2)$. Gerade bei größeren Datenmengen wird insbesondere das $|w|^2$ sehr schnell signifikant, wie der Tabelle 4.3.2 zu entnehmen ist.

Bei dem Einsatz der *Normalized Compression Distance* bzw. des eng verwandten *Compression-based Dissimilarity Measure* ist hingegen ein Aufwand von $\mathcal{O}(n^2c)$ zu erwarten, wobei c den Aufwand beschreibt, der maximal nötig ist einen Artikel zu komprimieren. Dieser Aufwand c ist, wie anhand der durchgeführten Tests erkennbar ist, signifikant geringer als die vergleichsweise teure Berechnung der k-Gramme im zuvor untersuchten Verfahren.

Für eine Analyse hinsichtlich der Qualität der Ähnlichkeitsbestimmung sei auf Abschnitt 4.2 (Textuelle Daten) verwiesen. Es wurde dabei deutlich, dass *NCD* bzw. *CDM* und *String kernel* vergleichbar gute Ergebnisse liefern.

Überführung struktureller Daten Damit graphenbasierte Clustering-Verfahren wie z.B. *Affinity Propagation* auf strukturelle Daten, die in Form von reellwertigen Vektoren oder Ähnlichkeitsbeziehungen vorliegen, angewendet werden können, muss eine Überführung der Daten in *gewichtete Netzgraphen* erfolgen. Hierbei werden aus Datenobjekten Knoten sowie aus deren Beziehungen (Abstände, Änhnlichkeiten) gewichtete Kanten. Bei der Wahl geeigneter Verfahren ist, neben der Qualität des Ergebnisses, auch die Laufzeit ein wichtiger Faktor. Es kann davon ausgegangen werden, dass Distanzen bzw. Ähnlichkeiten mit einem der zuvor genannten Verfahren bestimmt wurden und daher an dieser Stelle der Fokus auf die Überführung gelegt werden kann.

Das einfachste Verfahren dies zu bewerkstelligen ist die, in Abschnitt 2.5.1 vorgestellte, **Vollvernetzung** der Daten(-punkte). Hierbei entsteht ein Aufwand von $\mathcal{O}(n^2)$, da alle Daten paarweise verbunden werden. Der somit erzeugte Graph enthält daher $n(n-1)/2$ Kanten von denen die meisten Gewichtungen besitzen, die eine minimale Ähnlichkeit bzw. maximale Distanz repräsentieren. Solche Kanten sind für ein Clustering eher uninteressant und daher sollte, um den späteren Aufwand zu verringern, das Ziehen solcher Kanten vermieden werden.

Sinvoller ist also eine Reduktion der zu ziehenden Kanten auf relevante mit möglichst hoher Aussagekraft. Bereits an dieser Stelle kann somit der spätere Rechenaufwand der einzusetzenden Clustering-Verfahren reduziert werden, da diese lediglich bedeutungsstarke Kanten betrachten müssen.

Verfahren die einen solchen Aspekt berücksichtigen wurden in den Abschnitten 2.5.2 bis 2.5.5 vorgestellt. Da die Bestimmung von Nachbarschaftsgraphen mittels **ε-Neighborhood**, ***k*-Nearest Neighbor** und **Mutual k-Nearest**

Neighbor prinzipiell sehr ähnlich ablaufen und einen vergleichbaren Aufwand besitzen, soll im Folgenden nur auf die Komplexität bei der Erstellung eines kNN-Graphen eingegangen werden:

Aufwand für kNN_i bezüglich eines einzelnen Punktes i:

$$\begin{aligned} &\mathcal{O}(\texttt{add\_heap}(j)) + \mathcal{O}(\texttt{get\_heap}(k)) \ \forall j \\ &= \mathcal{O}(n \log n) + \mathcal{O}(k) \\ &= \mathcal{O}(n \log n) \end{aligned}$$

Für alle Punkte des kNN-Graphens:

$$\Rightarrow \mathcal{O}(n\, \mathcal{O}(\text{kNN}_i)) = \mathcal{O}(n^2 \log n)$$

Um für jeden Punkt seine k nächsten Nachbarn zu bestimmen, bietet es sich bei der Berechnung der Abstände an, die Distanzen in einer *heap*-Struktur zu hinterlegen. Hierbei entsteht ein Aufwand von $\mathcal{O}(\texttt{add\_heap}(j)) = \mathcal{O}(n \log n)$. Anschließend können die k nächsten Nachbarn mit einem Aufwand von $\mathcal{O}(k)$ bestimmt werden. Da diese Vorgänge für jeden der n Datenpunkte durchgeführt werden müssen, erhält man insgesamt eine Komplexität von $\mathcal{O}(n^2 \log n)$.

Diese drei Methoden lösen das Problem exakt, benötigen jedoch einen hohen Rechenaufwand. Eine Alternative dazu stellen *lokal sensitive Hash-Funktionen* dar, die k-Nearest Neighbor-Graphen approximativ erstellen können (Abschnitt 2.5.5). Der Vorteil liegt dabei in dem weitaus geringeren Aufwand von $\mathcal{O}(nlB)$ [KIW04] zur approximativen Suche der nächstgelegenen Punkte. Hierbei sei auf die Abbildung A.7 verwiesen, in der ein Vergleich zwischen approximativem und exaktem kNN-Graphen dargestellt ist. Wie in dem Bereich der roten Markierung zu erkennen ist, kann es bei einem approximativen Ansatz durchaus vorkommen, dass keine $k > 1$ Nachbarn von einem Punkt gefunden werden und daher kein perfekter kNN-Graph entsteht. Betrachtet man hingegen den gesamten approximierten Graphen und vergleicht ihn mit dem exakten, so ist ersichtlich, dass dieser, trotz einiger Abweichungen, seinem Vorbild hinreichend ähnlich ist.

Ergebnis Es lässt sich abschließend festhalten, dass es bei der Überführung von Datenpunkten in Graphen wichtig ist, überflüssige Kanten zu vermeiden, da für Algorithmen wie Affinity Propagation Graphen mit hoher Kantendichte einen zusätzlichen Mehraufwand bedeuten. Daher bieten approximative Verfahren interessante Ansätze, wenn es darum geht Rechenzeit einsparen zu können. Hier muss allerdings ein Kompromiss zwischen Geschwindigkeit und Genauigkeit des Ergebnisses getroffen werden.

Übersicht über unterstützende Verfahren

Verfahren	Eingabe	Ausgabe	Lokalität der Berechnung	Laufzeit-komplexität
Vollvernetzung	Vektoren	gewichtete Netzgraphen	lokal	$\mathcal{O}\left(n^2\right)$
kNN	Vektoren	gewichtete Netzgraphen	lokal	$\mathcal{O}\left(n^2 \log n\right)$
MkNN	Vektoren	gewichtete Netzgraphen	lokal	$\mathcal{O}\left(n^2 \log n\right)$
ε-Neighborhood	Vektoren	gewichtete Netzgraphen	lokal	$\mathcal{O}\left(n^2 \log n\right)$
approximatives kNN mittels LSH	Vektoren	gewichtete Netzgraphen	lokal	$\mathcal{O}\left(nlB\right)$
Normalized Compression Distance	textuelle Daten	Distanzen	lokal	$\mathcal{O}\left(n^2 c\right)$
Compression-based Dissimilarity Measure	textuelle Daten	Ähnlichkeiten	lokal	$\mathcal{O}\left(n^2 c\right)$
String Kernel	textuelle Daten	Ähnlichkeiten, Distanzen	lokal	$\mathcal{O}\left(n^2 k \lvert w\rvert^2\right)$

l : Anzahl verwendeter LSH-Funktionen
B: Bucketgröße
w: längster vorhandener Text
k : Substringlänge
c : Aufwand zur Kompression des längsten vorhandenen Textes

Tabelle 4.3.3: Einordnung unterstützender Verfahren

Bei der Bestimmung von Ähnlichkeiten bzw. Distanzen textueller Daten ist die Verwendung von Kompressions-Verfahren vorzuziehen. *String kernels*, die zwar eine hohe Genauigkeit bei entsprechend großer Substring-Länge aufweisen, stoßen hier schnell an die Grenzen der Berechenbarkeit. In dem zuvor beschriebenen Test mit Wikipedia-Artikeln zeigte sich schnell, dass bei bereits 192 KB Text ein kaum zu vertretender Zeitaufwand von mindestens 702 Sekunden (mit minimaler Genauigkeit) entsteht, wohingegen Kompressionsansätze nur 1,7 Sekunden benötigen.

Eine Einordnung und Gegenüberstellung der hier diskutierten Verfahren findet sich in Tabelle 4.3.3.

4.3.3 Kritische Betrachtung des CLDMs als Lösungsansatz

Durch das in Abschnitt 4.2.4 vorgestellte CLDM lässt sich effizient nach ähnlichen Ressourcen innerhalb eines zugrundeliegenden P2P-Systems, suchen. Effizient bedeutet hierbei, dass nicht relevante Datenobjekte frühzeitig ausgeschlossen werden können. Zudem ist zu erwarten, dass je genauer die Suchreferenz spezifiziert ist, desto mehr Ressourcen-Cluster ausgeschlossen werden können. Darüber hinaus lassen sich infrage kommende Datenobjekte schnell identifizieren, da in einem ersten Schritt lediglich deren Prototypen abgeglichen werden müssen. Somit wäre ein Abgleich in weniger als linearer Zeit ($< \mathcal{O}(n)$)

denkbar, sofern keine homogene Verteilung von Merkmalsausprägungen aller Ressourcen vorliegt.

Dieser Ansatz hat jedoch nicht nur Vorteile. Problematisch ist vor allem die zentrale Lösungsinstanz, die benötigt wird um die Cluster mit den jeweiligen Prototypen zu erstellen. Hierbei ist zwar klar geregelt, wer im Netzwerk diese Aufgabe inne hat, jedoch geht jegliche Kommunikation zum Aufbau und zur Modifikation des CLDMs durch die Cluster-Instanz. Dies kann gerade bei sensiblen Daten (von beispielsweise unterschiedlichen Kollaborationspartnern) sehr problematisch sein. Es muss nämlich eine zentrale Instanz gefunden werden, der alle Teilnehmer vertrauen, denn diese wird zwangsläufig alle Daten kennen, die in dem CLDM verfügbar sein sollen. Auch entsteht durch die CI ein „Flaschenhals“, wie er sonst eher bei Client-Server-Architekturen zu erwarten ist. Dies widerspricht jedoch dem eigentlichen Grundgedanken eines P2P-Netzwerkes.

4.3.4 Kritische Betrachtung des LSDMs als Lösungsansatz

Der zentrale Punkt des zweiten Datenmodells, welches in Abschnitt 4.2.7 in Form des Locality-Sensitive Data Models vorgestellt wurde, ist die Erweiterung eines P2P-Netzwerks um einen zusätzlichen Typ von Hashfunktionen. Ziel dieses Modells ist dabei – im Gegensatz zum CLDM – eine zentrale Instanz zur Verwaltung von Informationen über ähnliche Ressourcen zu vermeiden.

Durch den Einsatz von *lokal sensitiven Hash-Funktionen* ist es möglich, dass in linearer Zeit ($\mathcal{O}(nlB)$) die verfügbaren Ressourcen-IDs auf Buckets und diese auf entsprechende LS-Peers verteilt werden können. Mit den LS-Peers entstehen in dem P2P-Netzwerk Super-Peers, die die Verwaltung der Buckets übernehmen und dezentral organisiert sind. Da den LS-Peers allerdings nur Ressourcen-IDs bekannt sind (die z.B. mittels SHA-1 berechnet werden können), können sie keinerlei Rückschlüsse auf den Inhalt ziehen. Somit hat dieser Ansatz gerade in nicht vertrauenswürdigen Netzwerken Vorteile gegenüber einer „allwissenden“ Cluster-Instanz.

Ein weiterer betrachtenswerter Aspekt ist, dass der Berechnungsaufwand bei den einzelnen Peers liegt. Dieser Umstand ist bemerkenswert, da üblicherweise Clustering-Verfahren zentralisiert ablaufen und die betroffenen Ressourcen bekannt sein müssen. Bei der LSDM hingegen genügt es, dass jeder Peer seine Daten hasht und nur die IDs in Verbindung mit den Buckets an die LS-Peers überträgt. Auf diese Weise liegt der Rechenaufwand verteilt bei den Teilnehmern und die Super-Peers fungieren lediglich als „gelbe Seiten“.

Der für große n signifikante Geschwindigkeitsvorteil des LSH wird durch eine gewisse Ungenauigkeit erkauft, da es sich bei LSH um ein approximatives Verfahren handelt. Dies wird besonders deutlich, wenn man das Verfahren auf eine größere Datenmenge anwendet, und alle Datenpunkte, die von diesem als „ähnlich“ identifiziert wurden, farblich hervorhebt. In Abbildung A.6 sind zu diesem Zweck fünf „Datenwolken“ mit jeweils 100 Punkten dargestellt, die mittels Gauß-Verteilung zufällig erstellt wurden. Anschließend wurden LSH-Anfragen (*queries*) in den Zentren von vieren durchgeführt. Bei einer Durchführung von exakten kNN-Anfragen entstehen hierbei annähernd runde und gleichförmige Mengen an Nachbarn. Bei dem Einsatz von Locality-Sensitive Hashing entstehen jedoch aufgrund der Ungenauigkeit des Verfahrens asymmetrische Mengen, die in der Abbildung farbig hervorgehoben sind.

5 Verteilte Netzwerke mit Pools

Im Rahmen dieser Arbeit soll der Aufbau dezentraler Systeme, im Hinblick auf die durch Teilnehmer aufgespannten Netzwerkstrukturen, untersucht werden. Hierbei soll in diesem Kapitel speziell auf weit umspannende Netzwerke eingegangen werden, die aus lokalen Gruppierungen (Pools) von sich organisierenden Teilnehmern bestehen. Gerade solche, beispielsweise durch Globalisierung entstehende Netzwerke, bieten interessante Aspekte bezüglich einer kostenminimalen kollaborativen Produktentwicklung, die somit über große Distanzen abgewickelt werden muss.

5.1 Aufbau und Struktur

In diesem Abschnitt soll beschrieben werden, welche Kriterien für die zu untersuchenden Netzwerkstrukturen relevant sind. Zu diesem Zweck sei exemplarisch auf das in der Abbildung 1.1.2 dargestellte Anwendungsszenario verwiesen. Hierbei ging es darum, Bauteile bzw. Baugruppen ihrer Ähnlichkeit nach zu clustern und auf Pools derart aufzuteilen, dass jeweils genau *ein* Pool für ein Cluster verantwortlich ist. Kollaborationspartner in einem solchen Netzwerk sollten dadurch in der Lage sein, bei der Suche nach für sie relevanten Bauteilen frühzeitig nicht relevante Pools ausschließen zu können. Dies soll im Folgenden, mit dem Fokus auf den zugrunde liegenden Netzgraphen, verallgemeinert dargestellt werden. Wie der Abbildung 5.1.1 zu entnehmen ist, gliedern sich die, für diese Arbeit zu betrachtenden Netzwerke in *Pools*, die untereinander vernetzt Daten austauschen und beliebig weit voneinander entfernt sein können. Solche Pools lassen sich synonym, wie in Abbildung 5.1.2 dargestellt, als *Gemeinschaften* betrachten und wie folgt weiter untergliedern:

- Bei einer **Gemeinschaft** handelt es sich um eine feste, regional gebundene Struktur, in welcher sich Teilnehmer des Netzwerkes zu Gruppen zusammen finden können.
- Eine **Gruppe** wird durch den Zusammenschluss mehrerer Teilnehmer definiert, die ähnliche Ressourcen bereitstellen.

Um Aussagen darüber treffen zu können, wie gut ein Netzwerk geeignet ist, bestimmte Datenobjekte bezüglich ihrer Ähnlichkeiten sinnvoll gruppiert verwalten zu können, müssen einige Kriterien untersucht werden, die im Folgenden dargelegt werden sollen.

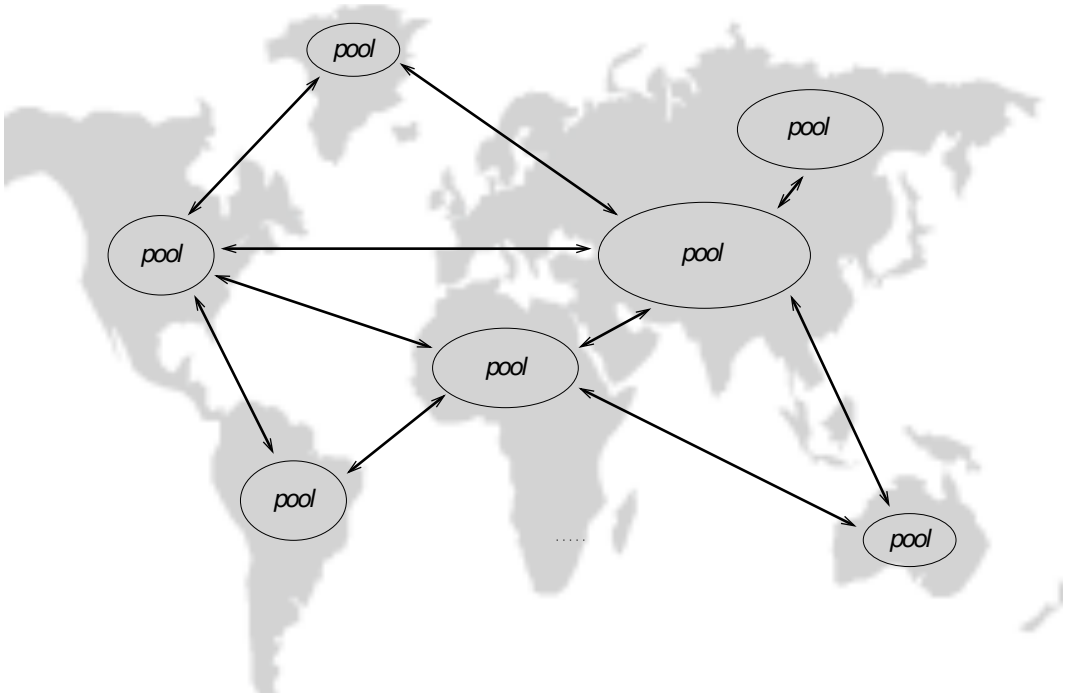


Abbildung 5.1.1: Verteilte Ressourcenverwaltung in Netzwerk-Pools [in Anlehnung an Kub+00, S.2]

Dichteverteilung Ein Netzgraph, wie er im Rahmen der Problemstellung dieser Arbeit zu betrachten sein soll, zeichnet sich dadurch aus, dass er viele kompakte Bereiche – quasi „Inseln“ – von Knoten mit hoher Kantendichte aufweist. Diese, als Pools bezeichneten Ballungen von Knoten, sind, geographisch betrachtet, dabei von weitestgehend leeren Bereichen umgeben. In den umgebenden Regionen verlaufen demnach nur wenige Kanten, die zudem mit vergleichsweise hohen Kantengewichten (beispielsweise Kosten) bewertet sein können. Es wäre also sinnvoll, Strategien zu entwerfen, wie solche dichten Bereiche in Graphen identifiziert werden können, um diesen gezielt bestimmte

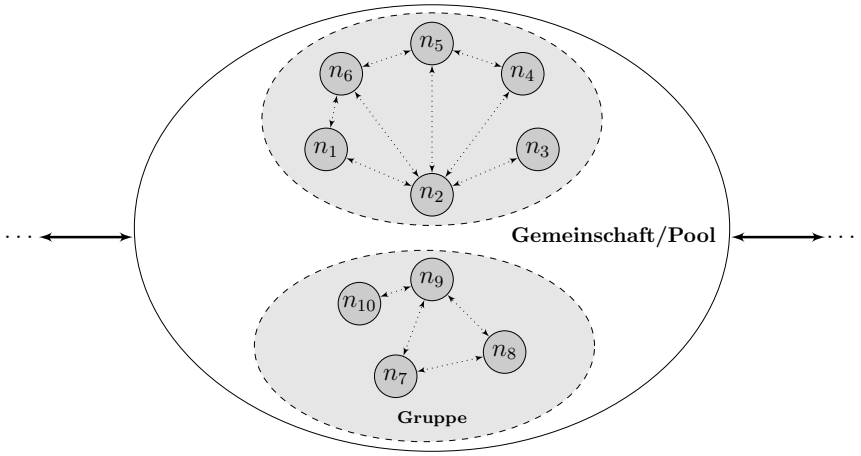


Abbildung 5.1.2: Ein Pool bestehend aus zwei Gruppen

Typen von Datenobjekten zuordnen zu können und zudem Suchkosten in solchen Netzen zu minimieren.

Gemeinschaften, Pools und Gruppen Gruppen, bestehend aus mehreren Knoten, können sich ungeplant bilden und erweitern, indem sich Teilnehmer zusammenfinden, oder neue zu bereits bestehenden hinzukommen. Darüber hinaus lassen sich mehrere Gruppen in Gemeinschaften zusammenfassen, die im weiteren, bei einer technischen Betrachtung von Netzwerken, als Pools bezeichnet werden. Legt der Begriff einer *Gemeinschaft* die Assoziation eines sozialen Geflechts nahe, in der eine Entität mit mehreren Gemeinschaften assoziiert sein kann, so soll doch im weiteren Verlauf dieser Arbeit davon ausgegangen werden, dass jeder Teilnehmer nur zu genau einer Gemeinschaft, bzw. Pool zugehörig ist und sich dies im Laufe seiner Partizipation an dem Netzwerk nicht ändern wird. Vielmehr soll der primäre Grund eines Teilnehmers, sich einer bestimmten Gemeinschaft anzuschließen, in seiner geographischen Lage liegen, und im Falle von mehreren, für ihn potentiell geeigneten Gruppen, derjenigen der Vorzug gegeben werden, deren verwaltete Ressourcen seinen am ähnlichsten sind. Ein solcher Vorzug bezüglich der in einer Gruppe angebotenen Daten, soll, wie im folgenden Abschnitt erläutert, durch gezieltes Befragen von sogenannten Repräsentanten geschehen.

Repräsentanten Haben sich Teilnehmer zu einer Gruppe innerhalb einer Gemeinschaft zusammengefunden, so soll einer von diesen als Repräsentant seiner Gruppe bestimmt werden können. Der Gedanke dabei ist, dass somit ein zentraler „Ansprechpartner“ in jeder Gruppe existiert, der externe Anfragen nach der Art der verwalteten Daten beantworten kann. Hierbei sollen von dem Repräsentanten Informationen über den Typ der in der Gruppe vorhandenen Ressourcen bereitgestellt werden. Des Weiteren soll, sofern dies für die hinterlegten Daten gleichen Typs möglich ist, ein „Durchschnitts“-Datum gebildet und durch den Repräsentanten verfügbar gemacht werden. Somit muss es bei der Suche nach ähnlichen Daten nicht mehr zwingend notwendig sein, *alle* Teilnehmer der Gruppen nach ihren Ressourcen zu fragen, sondern es kann bereits ausreichen zu wissen, wie die zu erwartenden Datenobjekte im „Durchschnitt“ aussehen, um vorweg nicht relevante Daten-Gruppen, oder ggf. sogar ganze Pools, ausschließen zu können.

Wie in Abbildung 5.1.3 abgebildet, soll der Knoten den Repräsentanten stellen, der in dem Subgraphen besonders exponiert ist. Die Identifikation kann beispielsweise aufgrund der verfügbaren Kapazitäten oder auch, wie hier abgebildet, schlicht durch den Knotengrad geschehen. Dieser Aspekt wird im

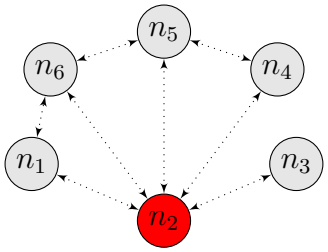


Abbildung 5.1.3: Eine Gemeinschaft mit Repräsentant (rot)

Abschnitt 5.6.3 näher untersucht.

Verbindungskosten Gerade wie in Abbildung 5.1.1 angedeutet, ist in globalen Netzwerken mit Datenübertragungen über weite Strecken und vielen Zwischenstationen zu rechnen. Hierbei lassen sich Kosten nach Kriterien wie Latenzen, verfügbaren Bandbreiten, Ausfallwahrscheinlichkeiten oder auch monetären Transitkosten (siehe Abschnitt 2.3.4: Infrastruktureigenschaften) unterscheiden.

Die Problematik lässt sich besonders gut am Beispiel von Latenzkosten erläutern: Sind Sender und Empfänger weit voneinander entfernt, das bedeutet zwischen beiden entsteht eine hohe Latenz, so macht sich dies bereits bei der Übertragung von mittelgroßen Daten (einige hundert Megabyte) bemerkbar. Dies liegt darin begründet, dass bei der Verwendung von Protokollen wie zum Beispiel Transmission Control Protocol / Internet Protocol (TCP/IP), seitens des Senders Daten in kleinere Segmente von wenigen Kilobyte (1500 Bytes bei Ethernet-Übertragungen) unterteilt und auf Bestätigungen vom Empfänger gewartet werden muss [Tan02, S. 409]. Somit kann sich latenzbedingt die Übertragungsrate stark vermindern und es zunehmend weniger attraktiv werden, Operationen in weit verteilten Netzwerken durchzuführen.

Im Rahmen dieser Arbeit soll bei der kostenoptimalen Ähnlichkeitssuche in weit verteilten Netzwerken davon ausgegangen werden, dass die einzelnen Pools bzw. Gruppen sehr kompakt und eine hohe Dichte aufweisen. Dies kann damit begründet werden, dass sich die einzelnen Gruppen von Teilnehmern, aufgrund ihrer geographischen Nachbarschaft, in regionalen Ballungen zusammenfinden. Die einzelnen Teilnehmer weisen daher einen sehr hohen Vernetzungsgrad auf, was als eine hohe *Intra-Pool-Dichte* bezeichnet werden kann. Im Gegensatz dazu existiert demnach in einem solchen Netzwerk eine vergleichbar geringe *Inter-Pool-Dichte*.

Bei der Betrachtung des Suchaufwandes, in den hier vorliegenden Netzwerken, lässt sich somit zum einen die Ebene des gesamten Netzes, also mit dem Fokus auf die Inter-Pool-Kosten, und zum anderen die Ebene der einzelnen

Pools (Intra-Pool-Kosten) untersuchen. Hierbei soll der Begriff der Kosten synonym mit dem zu erwartenden Aufwand verwendet werden, der notwendig ist, um Daten und Anfragen zwischen Teilnehmern auszutauschen.

5.2 Lokalisierungsablauf im Gesamtnetzwerk

Die Vorteile einer intelligenten Gruppierung von Ressourcen bezüglich ihrer Ähnlichkeit zueinander und entsprechender Aufteilung auf Pools zeigen sich bei der Betrachtung von Kosten, die bei der Suche nach ähnlichen Daten zu einem gegebenen Datum entstehen.

Für die Suche nach ähnlichen Ressourcen lassen sich drei unterschiedliche Ansätze festhalten die in den folgenden Abschnitten 5.2.1 bis 5.2.3 erläutert werden.

5.2.1 Naiver Ansatz

Der offensichtlich einfachste Ansatz ist es, wenn der suchende Teilnehmer seine Referenzspezifikation mit den Ressourcen aller übrigen Teilnehmer abgleicht. Hierbei wird sofort ersichtlich, dass dieses ein sehr ineffizientes Verfahren dar-

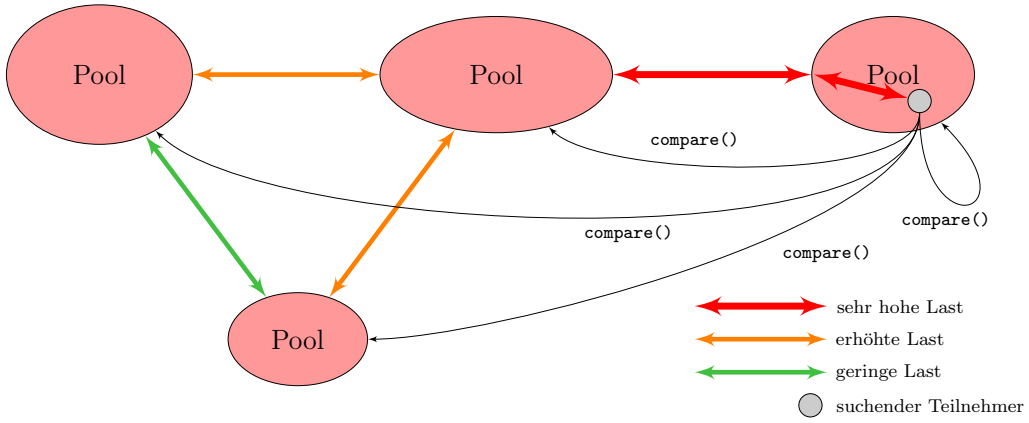


Abbildung 5.2.1: Naive Ähnlichkeitssuche

stellt: Egal ob der Suchende Teilnehmer seine Spezifikation herausgibt oder alle übrigen anfordert, es müssen insgesamt für n Teilnehmer n Nachrichten ausgetauscht werden. Somit entstehen Übertragungskosten, die sich aus der Summe über alle Inter-Pool-Pfade von dem Pool des Teilnehmers in alle Ziel-Pools und wieder zurück, sowie den Intra-Pool-Kosten, bilden.

In Abbildung 5.2.1 wird dieser Ansatz exemplarisch verdeutlicht. Ein Teilnehmer sucht im Netzwerk Ressourcen, die ähnlich zu einer von ihm gewählten Referenzspezifikation sein sollen. Hierbei vergleicht er mittels der angegebenen

Funktion `compare()` die in sämtlichen Pools vorhandenen Ressourcen mit seiner Spezifikation. Wie ersichtlich ist, kumuliert entlang der Netzwerk-Pfade zwischen ihm und den übrigen Pools eine zunehmende Last, da sämtliche Spezifikationen oder sogar Ressourcen als solche zu ihm transportiert werden müssen. Zudem entsteht in allen Pools eine erhöhte Netzwerk-Last (rot dargestellt), da schlimmstenfalls jeder Netzwerk-Teilnehmer Daten an den suchenden übertragen muss. Durch die generell vorhandenen hohen Latenzen und den Flaschenhals-Charakter zwischen den einzelnen Pools ist ein solcher Ansatz nicht realisierbar, insbesondere wenn davon ausgegangen werden kann, dass mehrere Teilnehmer simultan Suchvorgänge einleiten.

5.2.2 Befragung aller Repräsentanten

Eine Verbesserung stellt das Vorgehen dar, repräsentative Spezifikationen, für die jeweiligen Pools von den für diese verantwortlichen **Repräsentanten**, anzufordern. Hierbei muss bei einem Suchvorgang zunächst jeder Pool einmal angefragt werden, um aus den empfangenen Spezifikationen die vielversprechendste auszuwählen. Anschließend können innerhalb des gewählten Pools die Ressourcen gesucht werden, die zu der gewünschten Referenzspezifikation hinreichend ähnlich sind.

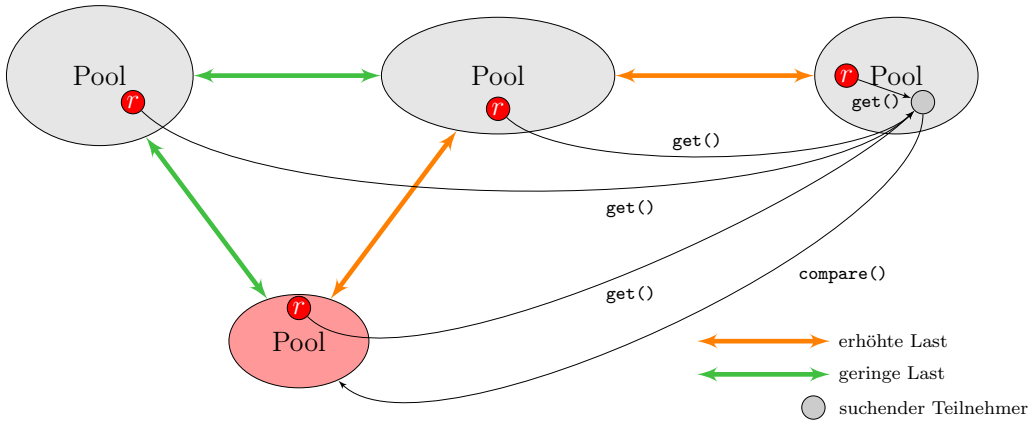


Abbildung 5.2.2: Repräsentantenbasierte Ähnlichkeitssuche

Wie in Abbildung 5.2.2 dargestellt, fragt ein Teilnehmer eines Pools zunächst alle Repräsentanten nach ihren Referenzspezifikationen und entscheidet, welche von diesen seiner gesuchten am ähnlichsten sind. In diesem Beispiel ist dies die Spezifikation eines einzelnen Pools. Somit kann der Suchende alle übrigen Pools vorweg ausschließen und die Suche auf den relevanten Pool beschränken. Daraus resultiert die dargestellte Orangefärbung der Verbindungskanten und des relevanten Ziel-Pools, welche eine erhöhte Last im Netzwerk illustrieren

soll. Hierbei kann, durch den Einsatz von vorgeschalteten Repräsentanten, eine hohe Last innerhalb nicht relevanter Pools vermieden werden. Auch muss nur für jeden Pool exakt eine Referenzspezifiaktion übertragen werden, welches in der Abbildung durch die `get()`-Operationen visualisiert wird. Anschließend kann vom suchenden Teilnehmer aus den vorliegenden Spezifikationen eine geeignete ausgewählt werden und mittels `compare()` innerhalb des relevanten Pools nach geeigneten Ressourcen gesucht werden.

5.2.3 Intelligentes Zwischenspeichern

Durch die Befragung von Repräsentanten lassen sich Kosten für die Befragung sämtlicher Teilnehmer innerhalb der Pools einsparen. Jedoch müssen nach wie vor für jeden Suchvorgang alle Repräsentanten in den Pools befragt werden. Die in Abbildung 5.2.3 dargestellte Erweiterung der *Befragung aller Repräsentanten* soll dessen Manko vermeiden. Dabei sollen, wie dargestellt, mittels `sync()`-Nachrichten die aktuellen Referenzspezifikationen zwischen allen Repräsentanten ausgetauscht werden, sodass diese stets über möglichst aktuelle Spezifikationen der übrigen Repräsentanten verfügen.

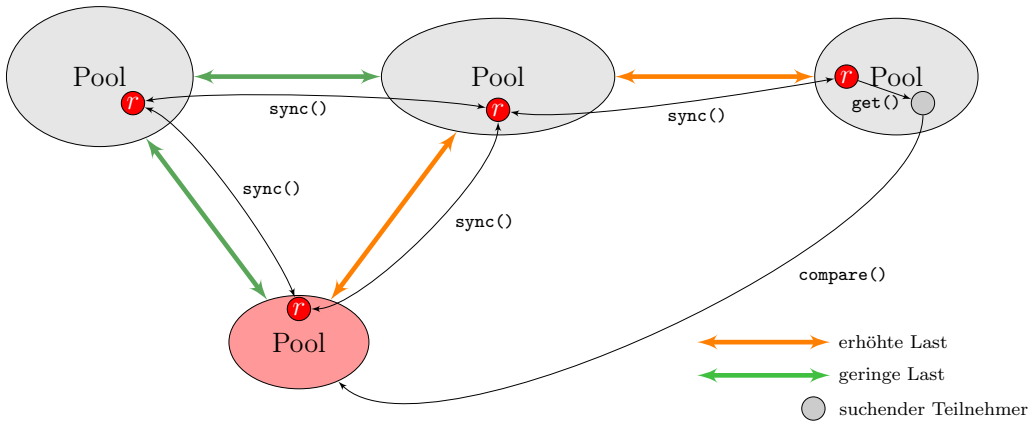


Abbildung 5.2.3: „Intelligentes Zwischenspeichern“ von Spezifikationen bei der repräsentantenbasierten Ähnlichkeitssuche

Ein `sync()`-Datenpaket soll dabei Zuordnungen von Spezifikationen zu Repräsentanten enthalten. Darüber hinaus besitzt jede Spezifikation einen Zeitstempel, anhand dessen ihre Aktualität festgehalten werden kann.

Für einen Teilnehmer genügt es somit zur Identifikation relevanter Pools, dass er seinen eigenen Repräsentanten nach Referenzspezifikationen befragt (`get()`), die dieser zuvor mittels `sync()` erhalten hat. Auf diese Weise muss ein suchender Teilnehmer nicht das gesamte Netzwerk mit Anfragen belasten und beschränkt sich somit auf seinen eigenen Pool.

Ein Austausch von `sync()`-Nachrichten soll dabei aufgrund der hohen Latenzen zwischen den Pools so übertragen werden, dass eine unnötige Belastung der jeweiligen Pools minimal ist. Daher wäre ein denkbarer Ansatz, dass jeder Pool entlang eines kostenminimalen Pfades seine Prototypen an die übrigen Pools verschickt. Jeder Empfänger aktualisiert daraufhin die ihm vorliegenden Spezifikationen, sofern die mitgeschickten Zeitstempel aktueller sind als die bereits vorhandenen, indem die vorhandenen Zuordnungen durch die neueren ersetzt werden. Des Weiteren könnten Pools, die auf einem solchen Übertragungspfad liegen, anhand der Spezifikationszuordnungen die weitergeleitet werden sollen, ihre eigenen Spezifikationslisten aktualisieren. Der genaue Ablauf ist in Abbildung A.8 (Anhang) dargestellt.

Dieser Ansatz minimiert zwar die Last, durch redundant auftretende Suchanfragen, jedoch kann, aufgrund der Latenzen zwischen den Pools, nicht garantiert werden, dass Änderungen der Referenzspezifikationen sofort zu allen Repräsentanten repliziert wurden.

Hat ein Teilnehmer Grund zur Annahme, dass die ihm vorliegenden Spezifikationen veraltet oder nicht aktuell genug sind, so bestehen **drei** Möglichkeiten in dieser Situation zu reagieren:

1. **Warten**
   Der Teilnehmer verhält sich passiv und wartet solange, bis bei seinem Repräsentanten aktualisierte Spezifikationen eintreffen. Hierbei muss sich der Teilnehmer natürlich sicher sein, dass es aktuellere Informationen im Netzwerk befinden, die noch repliziert werden müssen, da er ansonsten unbestimmt lange warten wird.

2. **Repräsentanten abfragen**
   Er wird selbst aktiv und fragt die aktuellen Spezifikationen selbst von allen Repräsentanten ab. Dieser Ansatz greift jedoch in die grundlegende Abkapselung der Funktionalität von Repräsentanten und gewöhnlichen Teilnehmern ein und ist daher nicht zu empfehlen. Zudem würden in diesem Fall vermutlich viele Teilnehmer einen solchen Weg wählen und somit das Netzwerk mit Anfragen fluten, was jedoch vermieden werden sollte.

3. **Re-`sync()` erzwingen**
   Die dritte Variante sieht vor, dass einem Repräsentanten mitgeteilt werden kann, nicht wie gehabt auf eingehende Aktualisierungen zu warten, sondern diese selbst bei allen übrigen Repräsentanten direkt abzufragen. Dieser Ansatz verletzt zwar nicht die Kapselung von deren Funktiona-

lität wie Punkt 2, allerdings bedeutet er einen größeren Übertragungsaufwand, wenn viele solcher Anfragen parallel statt finden. Aus diesem Grund sollte bei dem Einsatz dieses Verfahrens darauf geachtet werden, dass zwischen zwei re-`sync()`-Anfragen eine gewisse Mindestzeit vergeht.

Es existiert somit ein **Konflikt** zwischen *Aktualität* und *Geschwindigkeit* in der ein Ergebnis bezüglich der Suche nach relevanten Pools vorliegt.

5.3 Lokalisierungsablauf in Pools

Im vorherigen Abschnitt wurde die Suche nach ähnlichen Ressourcen und die Eingrenzung des Suchraumes auf relevante Pools mit der groben Sicht auf die Inter-Pool-Kosten bzw. -Nachrichtenübertragungen erläutert. Bei dem nun folgenden Fokus auf die Intra-Pool-Vorgänge zwischen den einzelnen Teilnehmern sollen die Abläufe innerhalb der Pools beschrieben werden, die notwendig sind, um Suchanfragen korrekt weiterzuleiten sowie verarbeiten zu können.

5.3.1 Passthrough: Weiterleitung von Anfragen und Daten durch Pools

Wie aus den Abbildungen 5.2.1 bis 5.2.3 ersichtlich wird, müssen Anfragen in Form von Vergleichsanfragen sowie Bauteilspezifikationen (oder allgemein Ressourcen) unter Umständen von Pools weitergereicht werden. Dies ist beispielsweise dann der Fall, wenn der kürzeste Pfad vom Sender zum Empfänger durch einen Pool verläuft und keine direkte Verbindung zwischen den Pools des Senders und des Empfängers liegt. In einer solchen Situation muss, durch die Gruppen innerhalb des Pools, eine entsprechende Weitergabe der Daten garantiert werden. Hierbei sind, wie in den folgenden Abbildungen 5.3.1 bis 5.3.3 dargestellt, drei Abläufe denkbar. Im schlimmsten Fall müssen eingehende Da-

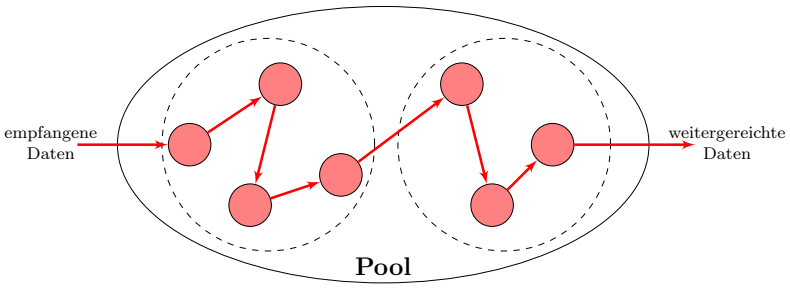


Abbildung 5.3.1: Übertragung von Daten durch Pools (vollständig)

ten von sämtlichen Teilnehmern weitergereicht werden (Abbildung 5.3.1), um den nächstfolgenden Pool, der auf dem Pfad zum Empfänger liegt, zu erreichen. Eine solche „Kette“ würde entstehen, wenn alle Teilnehmer auf einem Pfad angeordnet sind und dies dem kürzesten bzw. schnellsten Weg durch einen Pool

entspricht. Der Pool wird somit bei dieser Form der Weiterleitung *vollständig* in Anspruch genommen.

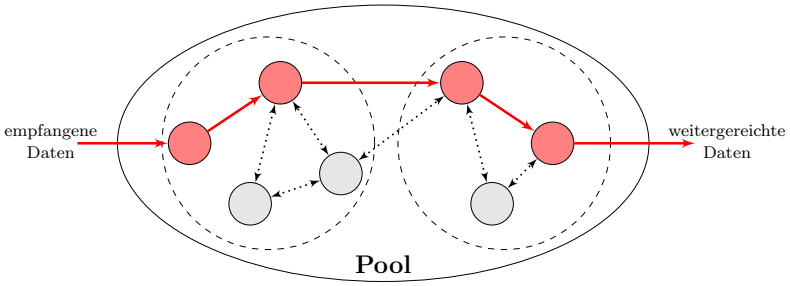


Abbildung 5.3.2: Übertragung von Daten durch Pools (partiell)

Da jedoch davon ausgegangen werden kann, dass die Vernetzung und die Kantendichte innerhalb einer Gemeinschaft relativ hoch ist, ist das folgende Szenario, dargestellt in der Abbildung 5.3.2, als realistischer anzusehen.

Die eingehenden Daten werden dabei von dem empfangsbereiten Eingangsteilnehmer auf dem Pfad mit den geringsten Übertragungskosten (z.B. Latenzen) an den Ausgangsteilnehmer weitergereicht und belasten den Pool bei der Übertragung nur *partiell*. Bezüglich der Bestimmung kürzester Pfadlängen sei auf Abschnitt 2.3.4 verwiesen. Eine dritte Möglichkeit stellt der Sonder-

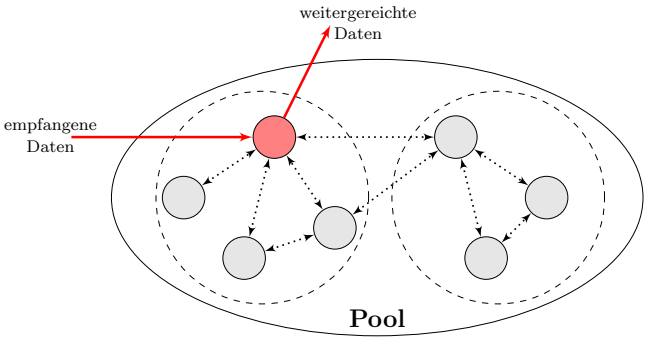


Abbildung 5.3.3: Übertragung von Daten durch Pools (peripher)

fall dar, in dem Eingangs- und Ausgangsteilnehmer dieselben sind. In einem solchen Fall kann von einer lediglich *peripheren* Inanspruchnahme des Pools zur Weiterleitung gesprochen werden, da nur ein einzelner Knoten für diese Aufgabe verantwortlich ist.

5.3.2 Anforderungen an Teilnehmer

Bei der gezielten Suche nach ähnlichen Ressourcen in Pools müssen von den einzelnen Teilnehmern bestimmte Funktionalitäten bereitgestellt werden, die hier näher erläutert werden. Bei der Übertragung von Daten soll dabei nach den, im vorherigen Abschnitt vorgestellten, Prinzipien verfahren werden.

Bereitstellung von Ressourcen Ein Teilnehmer muss über eine Funktion `get()` verfügen, mit der er Anfragen nach bestimmten Ressourcen beantworten kann, sofern diese Operation für den Anfragesteller erlaubt ist. Ist dies der Fall, so überträgt dieser die gewünschte Ressource, sofern vorhanden, an den Anfragenden.

Vergleich von Ressourcen Um die Möglichkeit zu schaffen, Ressourcen von einem entfernten Teilnehmer mit einer lokal vorliegenden Referenzspezifikation hinsichtlich bestimmter Anforderungen vergleichen zu können, muss eine Methode `compare()` existieren, mit der ein solcher Vergleich durchgeführt werden kann. Hierbei sind für den Teilnehmer, der ausgehend von seiner Referenzspezifikation Suchanfragen durchführt, zwei Ansätze denkbar:

1. Die Referenzspezifikation wird an alle infrage kommenden Teilnehmer aus den als relevant identifizierten Gruppen übertragen. Diese führen einen Vergleich durch und liefern Ergebnisse zurück, welche Aussagen über die Ähnlichkeiten bzw. Unterschiede zwischen der Referenzspezifikation und den von dem entfernten Teilnehmer verwalteten Ressourcen zulässt.

2. Es werden alle entfernten Teilnehmer die infrage kommen gebeten, ihre Ressourcenspezifikationen an den Suchenden zu übertragen, damit dieser lokal Vergleiche anstellen kann.

Bei dem zweiten Ansatz fällt auf, dass hierbei alle Teilnehmer die Spezifikationen der Ressourcen die sie verwalten übertragen müssen. Realistisch betrachtet kann davon ausgegangen werden, dass viele Teilnehmer mehr als nur eine Ressource verwalten. Somit würde, im Gegensatz zum ersten Ansatz, eine weitaus stärkere Asymmetrie im Bezug auf die Anzahl und Größe der zu übertragenden Daten für jede Vergleichsanfrage auftreten. Dies liegt darin begründet, dass jeder befragte Teilnehmer die vollständigen Spezifikationen jeder seiner Ressourcen übertragen müsste. In dem ersten Ansatz hingegen genügt es, lediglich die signifikant abweichenden Merkmalsausprägungen ($\Delta_{\texttt{Res}}$) zu übertragen, wodurch eine geringe benötigte Bandbreite und Latenz zu erwarten sind. Somit ist die zweite Variante eher von Nachteil und scheidet hinsichtlich des Ziels, eine minimale Belastung des Netzwerkes zu verursachen, aus.

5.3.3 Anforderungen an Repräsentanten

Da in jeder Gruppe ein Teilnehmer mit zusätzlicher Funktionalität existiert, der zum einen die Gruppe nach außen hin repräsentiert und zum anderen für

die Verwaltung des gruppenspezifischen Prototypen verantwortlich ist, müssen für diesen bestimmte Anforderungen gelten, die im Folgenden genauer spezifiziert werden.

Caching Um die Synchronisation von Prototyp-Spezifikationen, wie in Abbildung 5.2.3 dargestellt, realisieren zu können, muss jeder Repräsentant über eine entsprechende Caching-Funktionalität verfügen, um die Prototypen der übrigen Pools mit zugehörigen Zeitstempeln zu speichern und diese mit den übrigen Pools auszutauschen.

Bereitstellung von Prototypen Ähnlich wie ein gewöhnlicher Teilnehmer muss ein Repräsentant Daten zur Verfügung stellen. Hierbei handelt es sich allerdings nicht um gewöhnliche Ressourcen, sondern um Prototypen, die, wie in Abschnitt 4.2.2 definiert, einen „Durchschnittswert“ bezüglich aller verfügbaren Ressourcen in der jeweiligen Gruppe darstellen.

Gatekeeper zum Vergleich von Ressourcen Neben der bereits beschriebenen Möglichkeit Referenzspezifikationen direkt mit den Ressourcen der jeweiligen Teilnehmer abzugleichen, ist ein weiterer Ansatz denkbar, in dem der Repräsentant stellvertretend für seine Gruppe als *Proxy* Vergleichsanfragen entgegen nimmt und, wie in Abbildung 5.3.4, an die übrigen Teilnehmer weiterreicht. Hierbei wäre denkbar, dass der Repräsentant als *Gatekeeper* fungiert,

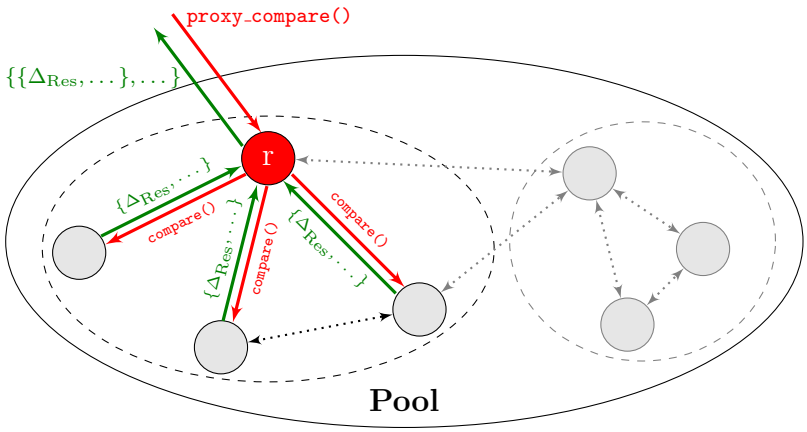


Abbildung 5.3.4: Proxy-Vergleich durch vorgeschalteten Repräsentanten

d.h. als „Schnittstelle“ zu den übrigen Gruppen und Pools entscheidet, welche Daten nach außen gelangen dürfen und welche nicht. Dies kann gerade in Gruppen, die sensible Daten verwalten, ein sinnvoller Ansatz sein.

Nach Abschluss der Vergleichs-Operationen liefern die Teilnehmer ihre Ergebnisse in Form von Mengen $\{\Delta_{\texttt{Res}}, \dots\}$, bestehend aus den Abweichungen

von Referenzspezifikation zu ihren Ressourcen ($\Delta_{\texttt{Res}}$), an den Repräsentanten zurück. Dieser leitet die Abweichungsmengen an den anfragenden Teilnehmer weiter.

Gatekeeper zur Bereitstellung von Ressourcen Analog zu der beschriebenen Funktionalität von `proxy_compare()` soll es neben Anfragen zum Vergleich auch möglich sein, Anfragen nach Ressourcen als solche verarbeiten zu können. Hierbei soll aus den gleichen Gründen ebenfalls der Repräsentant als *Gatekeeper* eingesetzt werden können. Diese Operation läuft im wesentlichen identisch mit dem zuvor beschriebenen Proxy-Vergleich ab mit dem Unterschied, dass anstelle von $\Delta_{\texttt{Res}}$ konkrete Ressourcen übertragen werden.

5.4 Zuordnung von Ressourcen auf Teilnehmer

Nachdem in den vorherigen Abschnitten die notwendigen Anforderungen an die Funktionalitäten von Teilnehmern und Repräsentanten (Abschnitt 5.1), sowie der generelle Ablauf zur Lokalisierung von relevanten Gruppen und ähnlichen Ressourcen (Abschnitt 5.2) dargelegt wurden, soll an dieser Stelle die noch ausstehende Zuordnung von Ressourcen auf Gruppen erläutert werden. Allgemein sollen Ressourcen gemäß ihrer Ähnlichkeit zueinander in Cluster, wie in Abbildung 5.4.1 exemplarisch dargestellt, zusammengefasst und alle Ressourcen, die sich innerhalb des *gleichen* Clusters befinden, auf die Teilnehmer *einer* Gruppe verteilt werden. Hierbei ist es durchaus denkbar, dass ein Pool für mehrere typengleiche Mengen von Ressourcen verantwortlich sein kann. Diese müssen dann auf die einzelnen Gruppen des Pools entsprechend aufgeteilt werden. Tritt jedoch der Fall ein, in dem ein Pool nur eine Gruppe enthält, aber mehr als ein Cluster verwalten soll, so bedeutet dies in der Konsequenz, dass unterschiedliche Cluster auf eine Gruppe verteilt werden müssen.

Für die Zuteilung von Ressourcen auf konkrete Teilnehmer sind dabei unterschiedliche Kriterien denkbar. Ein naiver Ansatz wäre dabei eine homogene Verteilung auf alle Peers der Gruppe, welche durch die Verwendung geeigneter Hash-Funktionen (beispielsweise SHA-1) realisiert werden kann. Gibt es jedoch signifikante Unterschiede in der Leistungsfähigkeit und dem Vernetzungsgrad der Peers, so ist es sinnvoll diesen Umstand mit zu berücksichtigen. Hierbei wäre es denkbar, Teilnehmern, die eine hohe Rechenkapazität aufweisen und mit vielen anderen Peers verbunden sind, mehr Ressourcen anzuvertrauen als anderen. Das Ziel sollte dabei sein, dass Ressourcen, die potentiell häufig angefragt werden, möglichst schnell, d.h. mit minimaler Latenz, übertragen zu können.

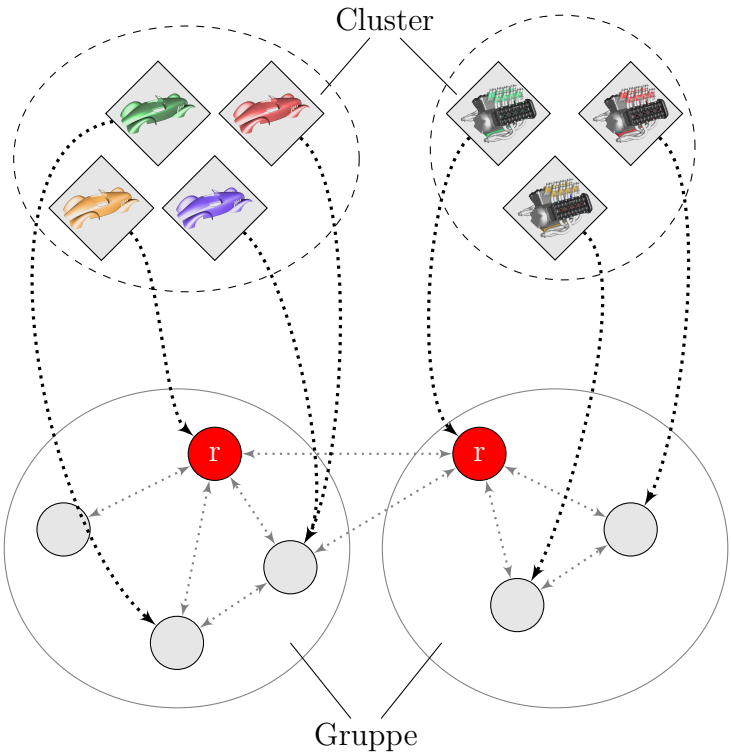


Abbildung 5.4.1: Zuordnung von Ressourcen-Clustern auf Teilnehmer in Gruppen

5.5 Abstraktion auf Netzgraphen

Ausgehend von dem Anwendungsszenario, welches im vorherigen Abschnitt vorgestellt wurde, soll im Folgenden die Problematik der graphentheoretischen Analyse und insbesondere der Erkennung von potentiellen Pools thematisiert werden. Bei der Betrachtung der für diese Arbeit relevanten Netzwerkstrukturen (vergleiche hierzu Abbildung 5.1.1, sowie Abbildung 5.1.2) ist eine Abstraktion auf allgemeine Netzgraphen notwendig. Ziel soll dabei sein, Aussagen bezüglich der Anzahl an Pools und die Anzahl der Teilnehmer in diesen treffen zu können, um in der Lage zu sein, Ressourcen-Cluster optimal auf Pools aufzuteilen.

5.5.1 Gewichtete Netzgraphen

Es ist offensichtlich, dass sich Teilnehmer als Knoten eines Netzgraphen darstellen lassen. Zur Modellierung von Kanten müssen jedoch vorab einige Feststellungen getroffen werden: Bezüglich des Daten- und Informationsflusses zwischen den Teilnehmern soll es keinerlei Priorisierung in eine bestimmte Kommunikationsrichtung geben. Das bedeutet, dass alle Eigenschaften (beispielsweise Abstände, Latenzen, Kosten) der Verbindungen zwischen zwei beliebigen Teilnehmern des Netzwerkes symmetrisch zu definieren sind. Es kann somit an-

genommen werden, dass wenn beispielsweise eine konstante Laufzeit von 5ms von Teilnehmer A nach Teilnehmer B existiert, diese ebenso für die entgegengesetzte Strecke von B nach A bei konstant 5ms liegt. Aus diesem Grund können die bisher dargestellten doppelt gerichteten Pfeile zwischen Teilnehmern in ungerichtete Kanten verallgemeinert werden. Relationsattribute zwischen Teilnehmern, die wie im oben genannten Beispiel in Form von Latenzen ausgedrückt werden können, lassen sich in Graphen als Kantengewichte zwischen Knoten modellieren. Zusammenfassend sei an dieser Stelle auf den Abschnitt 2.2.1 und im Besonderen auf die Definition 2.2.1 verwiesen, in dem die Eigenschaften von Netzgraphen erläutert werden, die im Folgenden relevant sind.

5.5.2 Sonderfall: ungewichtete Netzgraphen

Sollen Teilnehmer und deren Beziehungen untereinander auf Netzgraphen abstrahiert werden, so soll davon ausgegangen werden, dass sämtliche Verbindungen zwischen Teilnehmern untereinander und Pools bekannt sind. Jedoch ist der Fall denkbar, dass außer der Information über eine vorhandene Verbindung keine weiteren Beziehungsattribute, wie beispielsweise Übertragungskosten oder auch Latenzen (Abschnitt 5.1, Verbindungskosten), bekannt sind. Eine solche Situation kann eintreten, wenn Teilnehmer oder auch Pools diese Informationen, beispielsweise aufgrund von Vertraulichkeiten, geheim halten. Dies führt dazu, dass im Zweifel lediglich ungewichtete Netzgraphen vorliegen (Abschnitt 2.2.2), in denen definierte Abstandsmetriken wenig über tatsächliche Kosten aussagen. Aus diesen Gründen soll in Abschnitt 5.6.1 eine Möglichkeit vorgestellt werden, wie aufgrund vorliegender Kantendichten trotzdem entsprechende Gewichtungen bestimmt werden können, sowie in Abschnitt 5.6.2 ein Verfahren erläutert werden mit dem eine Identifizierung von Pools in ungewichteten Graphen ermöglicht wird.

5.6 Analyse von Netzgraphen

Basierend auf der Abstraktion von verteilten Netzwerken auf gewichtete oder ungewichtete Netzgraphen (Abschnitt 5.5) soll nun untersucht werden, wie Gemeinschaften identifiziert und Repräsentanten bestimmt werden können. Das Ziel soll dabei sein,

- die Anzahl an Pools zu erkennen,
- jeden Teilnehmer einer Gruppe bzw. einem Pool zuzuordnen,

- aus jeder Gruppe einen Teilnehmer als Repräsentanten zu bestimmen sowie
- Kantengewichte, sofern sie nicht vorhanden sind, aus der Netzstruktur abzuleiten.

5.6.1 Berechnung von Kantengewichten

Liegen, wie in Abschnitt 2.6 beschrieben, Netzgraphen vor, in denen **keine oder nur teilweise Kantengewichte** vorhanden sind, so bieten sich zwei Ansätze an, um Aussagen über vorhandene Gemeinschaften in diesen Netzen treffen zu können. Dies kann entweder durch den Einsatz von Algorithmen erreicht werden, die auf ungewichtete Netzgraphen anwendbar sind (siehe Tabelle 4.3.1) oder durch eine Vorverarbeitung der Graphen um Kantengewichte zu berechnen.

Eine solche Vorverarbeitung dient dabei, wie in Abbildung 4.2.6 dargestellt, zur Überführung der vorliegenden Datenstruktur in Form von ungewichteten Netzgraphen in gewichtete. Hierbei bietet sich das Verfahren der **Common Neighborhood Subgraph Density** (siehe Abschnitt 2.6.1) an.

Auf Basis der Knotengrade innerhalb des Graphens lassen sich hiermit Gewichtungen zwischen Knoten eines Netzes bestimmen, die einen Indikator dafür darstellen, inwiefern zwei Teilnehmer zu einer Gemeinschaft gehören (siehe Gleichung (2.6.3)). Je größer dieser Wert ausfällt, desto höher ist der Grad ihrer Interaktion und somit auch ihre Gemeinschaftszugehörigkeit zueinander. Die CND berechnet folglich Ähnlichkeiten, die im Folgenden Abschnitt zur Bestimmung von Gemeinschaften von Bedeutung sind.

5.6.2 Identifikation von Gemeinschaften

Zur Erkennung von Gemeinschaften in Netzgraphen bieten sich graphenbasierte Clustering-Verfahren an, die in der Lage sind (gewichtete) Adjazenzmatrizen zu verarbeiten und hieraus Knoten in Cluster aufzuteilen. Wie bereits in Abschnitt 4.2.3 dargelegt, müssen die einzusetzenden Algorithmen zudem Repräsentanten bestimmen können, die dann die in Abschnitt 5.1 vorgestellten Aufgaben übernehmen sollen.

Affinity Propagation Ein Verfahren, welches hierbei besonders geeignet ist, ist die *Affinity Propagation*. Die Gründe liegen nicht nur in der Fähigkeit eine Vielzahl an Datenstrukturen verarbeiten zu können – hierzu zählen reellwertige Datenpunkte sowie Ähnlichkeits- und Distanzmatrizen – sondern auch darin

Repräsentanten (hier Exemplare genannt) zu erhalten. Da der Ablauf darin besteht, dass *reellwertige Nachrichten zwischen den Datenpunkten ausgetauscht werden* [FD07a, S. 972], kann es sich zudem anbieten den Prozess der Exemplarfindung und Clusterzuordnung nicht von einer einzelnen Cluster-Instanz durchzuführen, sondern zu verteilen. Da Frey und Dueck [FD07a] zur plastischen Darstellung des Verfahrens ohnehin von einem Nachrichtenaustausch sprechen, bietet sich die Überlegung an, ob die vorhandenen Knoten (Teilnehmer) diese Aufgabe übernehmen könnten. Dies erscheint gerade in Hinblick auf die zugrundeliegenden, verteilten Umgebungen – die für diese Arbeit im Vordergrund stehen – ein potenziell lohnenswertes Ziel zu sein, um den Cluster-Aufwand zu verteilen. Auch kann unter Umständen der Aufwand in Netzwerken mit erhöhter Volatilität bezüglich der angemeldeten Teilnehmer weiter reduziert werden, da mitunter nicht alle Teilnehmer erneut Nachrichten austauschen müssen, bloß weil in einem entlegenen Bereich des Netzwerkes Knoten wegfallen bzw. hinzukommen.

Ant Colony Optimization Offensichtlich bietet sich dieser Ansatz, der in Abschnitt 2.7.4 vorgestellt und im Rahmen der Evaluation von Kapitel 4 bereits betrachtet wurde, für eine Realisierung mittels **Agenten** an, da die einzelnen *Ameisen* selbstständig eine mögliche Lösung für eine Zweiteilung des (Teil-) Graphens bestimmen. Es wird jedoch ein zentraler Agent benötigt (*Queen*), der für die Zuteilung von Ameisenagenten auf Teilgraphen sorgen muss. Zudem muss die Queen die lokal optimalen Lösungen empfangen, anhand dieser eine global optimale Lösung wählen und die Pheromonbelegungen aktualisieren.

Im Rahmen dieser Arbeit wurden zwei agentenbasierte Varianten der Ant Colony Optimization entwickelt (*zentrale* sowie *hybride ACO*) und mit Hilfe der Agentenplattform *SPADE* (siehe [PA12a] und [PA12b]) implementiert.

- **Zentrale ACO** Bei der zentralisierten Variante existiert eine Queen, die eine alleinige Zuordnung der Ameisen auf den zu untersuchenden (Teil-) Graphen festlegt. Zu diesem Zweck bestimmt die Queen zuerst, welche Agenten als Ameisen eingesetzt werden sollen und übermittelt diesen die notwendigen Informationen über die Graphenstruktur sowie die initiale Pheromonbewertung und die gewünschte Anzahl an iterationen.

  Die Ameisen bewerten nun iterativ den Graphen und übermitteln ihre lokalen Optima an die Queen. Diese startet anschließend eine neue Iteration mit den aktualisierten Pheromonwerten und dem neuen globalen Optimum.

Ist eine optimale Zweiteilung gefunden, so werden von der Queen erneut Ameisen auf die beiden Subgraphen aufgeteilt und der Ablauf beginnt von vorne.

- **Hybride ACO** Wie in dem zentralen Ansatz ersichtlich ist, verläuft jeglicher Nachrichtenaustausch über genau eine Queen und betrifft darüber hinaus auch alle Rekursionsebenen. Aus diesem Grund entsteht ein „Bottleneck“ mit negativen Auswirkungen auf die Effizienz des Verfahrens.

  Es ist daher sinnvoller die Last durch den Einsatz mehrerer Queens zu verteilen. Dies geschieht, indem eine **oberste Queen** eingesetzt wird und bei jedem Rekursionsschritt jeweils eine Ameise, die für einen Teilgraphen verantwortlich ist, die Rolle einer **untergeordneten Queen** übernimmt. Somit können die einzelnen Subgraphen unabhängig voneinander bearbeitet werden und anschließend die Ergebnisse (entlang des Rekursionsbaumes) nach oben zur obersten Queen übermittelt werden.

Der Quellcode für die zentrale sowie hybride Lösung befindet sich auf der CD, die dieser Arbeit beigelegt ist.

5.6.3 Identifikation von Repräsentanten

Wurden Gemeinschaften in Netzgraphen identifiziert, so müssen anschließend Repräsentanten bestimmt werden. Dieser Schritt ist notwendig, damit die in Abschnitt 5.3.3 spezifizierten Anforderungen von dem Netzwerk erfüllt werden können. Für diese Aufgabe gibt es mehrere mögliche Ansätze, von denen im Folgenden ausgewählte erläutert werden.

Knotengrad Ein möglicher Ansatz ist die Bestimmung eines Repräsentanten anhand von Knotengraden. Hierbei werden, wie in der Abbildung 5.6.1 durch

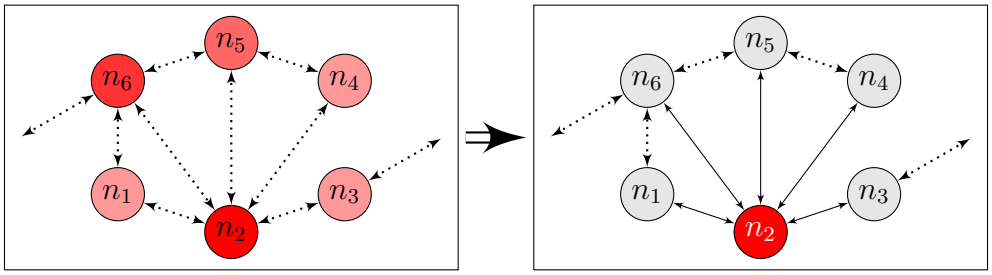


Abbildung 5.6.1: Bestimmung eines Repräsentanten anhand von Knotengraden

unterschiedliche Rotfärbungen dargestellt, Knoten entsprechend ihres Grades gewichtet. Der Knoten mit der stärksten Färbung bzw. höchsten Gewichtung (hier n_2) wird dann zum Repräsentanten ernannt.

Kantengewichte Denkbar ist aber auch eine Methode, die Kantengewichte (z.B. Übertragungskosten) mit einbezieht. Wie in Abbildung 5.6.2 zu erkennen

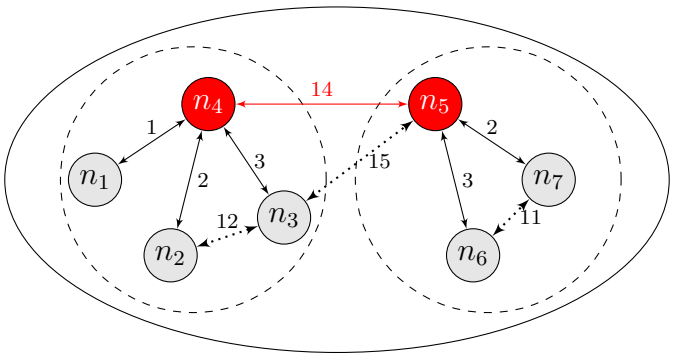


Abbildung 5.6.2: Bestimmung von Repräsentanten anhand von Kantengewichten

ist, wurden die Knoten zu Repräsentanten ernannt, deren Kantengewichte zu den übrigen Knoten ihrer Gruppe minimal sind. Ein solcher Ansatz hätte den Vorteil, dass beispielsweise bei `proxy`-Vergleichen minimale Übertragungskosten anfallen würden.

Affinity Propagation Die dritte Alternative nutzt den Umstand, dass bei dem Verfahren der Affinity Propagation, durch den Austausch von *responsibility* und *availability*-Nachrichten bereits bei der Identifikation von Gemeinschaften sogenannte Exemplare bestimmt werden. Exemplare haben die Eigenschaft, dass sie von allen Knoten ihres Clusters als optimale Vertreter ihrer Gemeinschaft (anhand von Kantengewichten) identifiziert wurden. Somit erfüllen Exemplare die gleichen Eigenschaften die für einen Repräsentanten benötigt werden und können als ein solcher festgelegt werden. Der Vorteil hierbei liegt in der Tatsache, dass somit kein separates Verfahren durchgeführt werden muss.

5.7 Evaluation

In diesem Abschnitt sollen die in Abschnitt 5.2 vorgestellten Suchverfahren evaluiert werden. Dabei wird eine Kostenanalyse hinsichtlich der *naiven* und *Repräsentantenbasierten* Suche sowie dem *intelligenten Zwischenspeichern* von Prototypen durchgeführt. Zudem soll die Identifikation von Gemeinschaften und Repräsentanten (Abschnitt 5.6.2 und Abschnitt 5.6.3) bewertet werden.

5.7.1 Kosten der Lokalisierung von Ressourcen

Bei der Bewertung der Kosten zur Suche ähnlicher Ressourcen ist anzunehmen, dass die *Intra*-Pool-Kosten im Vergleich zu den *Inter*-Pool-Kosten, aufgrund der weitaus höheren Vernetzungsdichte und kompakteren Struktur innerhalb

der Pools, zu vernachlässigen sind. Aus diesem Grund werden an dieser Stelle (bei der Betrachtung des gesamten Netzwerks) die Kosten innerhalb der Pools pauschal mit einem Wert $\leq \varepsilon$ (mit $\varepsilon > 0$) bewertet.

Bezeichnungen	
P_A	Ausgangspool (in dem sich der Teilnehmer befindet, der Suchanfragen stellt)
PE	Menge an entfernten Pools, d.h. alle Pools außer P_A
PZ	Menge an Zielpools die potenziell für Suchanfragen geeignet sind
P_Z	Zielpool
Pools	Menge vorhandener Pools ($PE \cup \{P_A\}$)
$\mathcal{W}^{\min}_{A,P}$	kostenminimaler Pfad zwischen Ausgangspool A und Zielpool P

Kosten	
K_{naiv}	Kosten der naiven Suche
K_{repr}	Kosten der Suche mittels Repräsentanten
K_{sync}	Kosten zur Synchronisation von Prototyplisten zwischen Repräsentanten
K_{cache}	Kosten der Suche mittels intelligentem Zwischenspeichern
$k(P_A, P_Z)$	Übertragungskosten zwischen Ausgangspool und Zielpool
$k(P_A, P)$	Übertragungskosten zwischen Ausgangspool und generellem Pool
$k(i, j)$	Übertragungskosten entlang der Kante (i, j) (Inter-Pool-Kosten)
ε	Intra-Pool-Kosten

Tabelle 5.7.1: Bezeichnungen und Kostentypen bei der Ressourcenlokalisation

Naiver Ansatz Formal ausgedrückt lässt sich die naive Suche aus Abschnitt 5.2.1 – ausgehend von einem Ausgangspool P_A und einer Menge an entfernten Pools PE – als folgende Kostenfunktion K_{naiv} darstellen:

$$K_{\text{naiv}}(P_A) = \underbrace{(|P_A| - 1) \cdot 2\varepsilon}_{\substack{\text{Suchkosten im} \\ \text{Ausgangspool}}} + \underbrace{2 \sum_{P \in PE} |P| \cdot [k(P_A, P) + 2\varepsilon]}_{\text{Suchkosten in den entfernten Pools}}$$

Bei dieser Art der Suche wird der unbeschränkte Suchraum vollständig abgesucht. Kosten entstehen somit für den Suchvorgang im eigenen Pool (Ausgangspool P_A) sowie um alle Teilnehmer in den übrigen Pools (Zielpools PZ) zu befragen. Die Funktion $k(P_A, P)$ gibt dabei die Kosten an, die für die Übertragung einer Spezifikation auf dem kürzesten Pfad (siehe Abschnitt 2.3.4, Gleichung (2.3.22)) vom Pool P_A nach P entstehen. Die Menge an entfernten

Pools PE umfasst alle übrigen Pools, zu denen der Suchende Teilnehmer nicht gehört.

Befragung aller Repräsentanten Zur Verringerung der Suchkosten wurde in Abschnitt 5.2.2 der Einsatz von Repräsentanten vorgestellt. Die Kostenfunktion lässt sich dabei wie folgt aufstellen:

$$K_{\text{repr}}(P_A) = \underbrace{2\varepsilon}_{1.} + \underbrace{2 \sum_{P \in PE} \Big(k(P_A, P) + 2\varepsilon \Big)}_{2.} + \underbrace{2(|P_A| - 1)\varepsilon}_{3.} + \underbrace{2 \sum_{P_Z \in PZ} \Big[|P_Z| \cdot (k(P_A, P_Z) + 2\varepsilon) \Big]}_{4.}$$

Diese besteht aus vier unterschiedlichen Teilkosten die bei dem Suchvorgang betrachtet werden müssen.

1. **Prototypen vom eigenen Repräsentanten beziehen** Den geringsten Aufwand stellt die Abfrage des Prototyps vom eigenen Repräsentanten dar, um entscheiden zu können, ob sich die gesuchten Ressourcen im eigenen Pool befindet. Hier muss lediglich eine Anfrage innerhalb des Pools gesendet und der Prototyp empfangen werden, wodurch zu vernachlässigende Kosten von 2ε (ε für Anfrage und anschließend ε für Antwort) entstehen.

2. **Prototypen aus den entfernten Pools beziehen** Um die Repräsentanten in den übrigen Pools befragen zu können, ist ein höherer Aufwand als bei (1.) nötig. Es kommen hierbei im wesentlichen die hohen Kosten zum tragen, die bei der Übertragung von Anfragen aus dem Ausgangspool P_A in die entfernten Pools PE anfallen.

3. **Suche in eigenem Pool** Passt der Prototyp des eigenen Repräsentanten zu der Suchanfrage, so kommt der lokale Pool für die Suche infrage. Es entstehen daher Kosten für die Befragung aller Teilnehmer, die sich mit in dem lokalen Pool befinden. Ist dieser Pool jedoch nicht relevant, so entfällt dieser Term.

4. **Suche in den Zielpools** Abschließend soll in allen Zielpools $PZ \subseteq PE$, die nach Abgleich ihrer Prototypen relevant sind, gesucht werden. Hierbei müssen für alle Teilnehmer in den entfernten Zielpools P_Z Transferkosten in Höhe von $|P_Z| \cdot k(P_A, P_Z)$ berücksichtigt werden.

Intelligentes Zwischenspeichern Dieser Ansatz wurde in Abschnitt 5.2.3 spezifiziert. Hierbei ist davon auszugehen, dass sich der Transferaufwand von `sync()`-Nachrichten in Grenzen hält, da mit maximal einem Repräsentanten in jeder Gruppe das Verhältnis im Vergleich zu gewöhnlichen Teilnehmern sehr gering ist. Des Weiteren ist eine erneute `sync()`-Übertragung auch nur dann notwendig, wenn sich der „Durchschnittswert“ der Ressourcen innerhalb eines Pools derart ändert, so dass eine modifizierte Referenzspezifikation erstellt und zwischen den Pools propagiert werden muss. Für den maximal zu erwartenden Aufwand kann davon ausgegangen werden, dass alle Spezifikationen ausgetauscht werden müssen. Findet eine, wie in Abschnitt 5.2.3 vorgeschlagene und in Abbildung A.8 dargestellte, Verteilung von `sync`-Nachrichten statt, indem Nachrichten auf kostenminimalen Pfaden $\mathcal{W}_{A,P}^{\min}$ zwischen dem Ausgangspool A und dem Zielpool P übertragen werden, so entstehen Kosten, die durch K_{sync} beschrieben werden. Hierbei müssen schlimmstenfalls zwischen allen Pools Nachrichten ausgetauscht werden. Für die Bestimmung der kostenminimalen Pfade bietet sich das Verfahren von *Dijkstra* an, welches in Abschnitt 2.3.4 erläutert und formal im Algorithmus A.2 abgebildet ist.

$$K_{\text{cache}} = 2\varepsilon + \underbrace{2 \sum_{P_Z \in PZ} \Big[|P_Z| \cdot (k(P_A, P_Z) + 2\varepsilon) \Big]}_{\text{Suche in den Zielpools } PZ}$$

$$K_{\text{sync}} = \underbrace{\sum_{A \in \text{Pools}} \sum_{P \in PZ} \sum_{(i,j) \in \mathcal{W}_{A,P}^{\min}} \Big(k(i,j) + 2\varepsilon \Big)}_{\substack{\text{Kosten bei einem Routing} \\ \text{über kostenminimale Pfade } \mathcal{W}_{A,P}^{\min}}}$$

Ein Teilnehmer, der eine Suchanfrage stellt, kann bei diesem Ansatz seinen Repräsentanten direkt nach relevanten Pools befragen (sofern dieser über aktuelle Prototypen der entfernten Pools verfügt) und anschließend, wie in K_{cache} dargestellt, eine Suche in den Zielpools durchführen.

Kostenvergleich Für den Vergleich der Kosten sollen, wie eingangs erläutert, Intra-Pool-Kosten ε als vernachlässigbar betrachtet werden. Die daher relevanten Kosten sind im Folgenden aufgeführt:

$$K_{\text{naiv}}^{\text{relevant}}(P_A) = 2 \sum_{P \in PE} |P| \cdot k(P_A, P)$$

$$K_{\text{repr}}^{\text{relevant}}(P_A) = \underbrace{2 \sum_{P \in PE} k(P_A, P)}_{\substack{\text{Prototypen aus den ent-} \\ \text{fernten Pools beziehen}}} + \underbrace{2 \sum_{P_Z \in PZ} |P_Z| \cdot k(P_A, P_Z)}_{\text{Suche in den Zielpools}}$$

$$K_{\text{cache}}^{\text{relevant}}(P_A) = \underbrace{2 \sum_{P_Z \in PZ} |P_Z| \cdot k(P_A, P_Z)}_{\text{Suche in den Zielpools } PZ}$$

$$K_{\text{sync}}^{\text{relevant}} = \underbrace{\sum_{A \in \text{Pools}} \sum_{P \in PZ} \sum_{(i,j) \in \mathcal{W}_{A,P}^{\min}} k(i,j)}_{\substack{\text{Kosten bei einem Routing} \\ \text{über kostenminimale Pfade } \mathcal{W}_{A,P}^{\min}}}$$

Es ist erkennbar, dass bei dem Einsatz der naiven Suche die relevanten Kosten ($K_{\text{naiv}}^{\text{relevant}}$) größer sind als bei der Verwendung von Repräsentanten ($K_{\text{repr}}^{\text{relevant}}$), sofern insgesamt mehr als zwei Pools existieren und mindestens ein Pool ausgeschlossen werden kann. In Fällen, in denen kein Pool ausgeschlossen werden kann, würden die Kosten bei $K_{\text{repr}}^{\text{relevant}}$ sogar höher sein, da initial die Prototypen aus den entfernten Pools bezogen werden müssen.

Die Suche mit Hilfe des intelligenten Zwischenspeicherns von Prototypen verursacht hingegen relevante Kosten ($K_{\text{cache}}^{\text{relevant}}$), die lediglich aus der Suche in relevanten Pools bestehen. Somit ist dieser Ansatz kostengünstiger als $K_{\text{repr}}^{\text{relevant}}$. Der Grund liegt vorallem darin, dass die initiale Beschaffung von Prototypen entfällt. Allerdings entfallen diese Kosten nicht vollständig, da hierbei Kosten bei den Repräsentanten entstehen ($K_{\text{sync}}^{\text{relevant}}$). Diese müssen mittels `sync()`-Nachrichten die Informationen über Prototypen aktuell halten.

Zusammenfassend lässt sich festhalten, dass die Varianten mittels Repräsentanten sowie Zwischenspeicherung nur dann sinnvoll eingesetzt werden können, wenn ein Netzwerk vorliegt, welches aus mindestens zwei Pools besteht und bei Suchanfragen mindestens ein Pool ausgeschlossen werden kann.

Anwendungsbeispiel Um die relevanten Kosten, die bei den jeweiligen Suchansätzen anfallen, mit konkreten Werten gegenüberzustellen, wurden diese anhand eines Anwendungsszenarios berechnet. Es soll in dem Beispiel davon ausgegangen werden, dass in einem Netzwerk insgesamt 16 Pools mit jeweils 100 Teilnehmern existieren. Zwischen allen Pools bestehen Übertragungskosten $k(i, j)$ von 10 Einheiten. Zudem ist – ausgehend von einem suchenden Teilnehmer – die Menge an relevanten Zielpools PZ eine Teilmenge aller entfernten Pools, d.h. der lokale Ausgangspool soll als relevanter Zielpool nicht infrage kommen.

Es sollen nun die Kosten für die Fälle $|PZ| = 1$ und $|PZ| = 15$ betrachtet werden, d.h. für die Fälle in denen *einer* sowie *alle* entfernten Pools infrage kommen.

Aus diesen Bedingungen lassen sich die einzelnen relevanten Kosten wie folgt berechnen:

- **Naive Suche**

Für $|PZ| = 1$ **und** $|PZ| = 15$ gilt:

$$K_{\text{naiv}}^{\text{relevant}} = 2 \cdot 15(100 \cdot 10) = 30.000$$

- **Repräsentantenbasierte Suche**

Für $|PZ| = 1$ gilt:

$$\begin{aligned} K_{\text{repr}}^{\text{relevant}} &= 2 \cdot 15 \cdot 10 + 2 \cdot (100 \cdot 10) \\ &= 2.300 \end{aligned}$$

Für $|PZ| = 15$ gilt:

$$\begin{aligned} K_{\text{repr}}^{\text{relevant}} &= 2 \cdot 15 \cdot 10 + 2 \cdot 15(100 \cdot 10) \\ &= 30.300 \end{aligned}$$

- **Caching-basierte Suche**

Für $|PZ| = 1$ gilt:

$$\begin{aligned} K_{\text{sync}}^{\text{relevant}} &= (16 \cdot 15) \cdot 10 = 2.400 \\ K_{\text{cache}}^{\text{relevant}} &= 2 \cdot (100 \cdot 10) = 2.000 \end{aligned}$$

Für $|PZ| = 15$ gilt:

$$\begin{aligned} K_{\text{sync}}^{\text{relevant}} &= (16 \cdot 15) \cdot 10 = 2.400 \\ K_{\text{cache}}^{\text{relevant}} &= 2 \cdot 15(100 \cdot 10) = 30.000 \end{aligned}$$

Zusammenfassend sind die Kosten in der Tabelle 5.7.2 dargestellt. Es wird hierbei deutlich sichtbar, dass bei Suchanfragen in denen alle entfernten Pools infrage kommen, keine Kosteneinsparung gegenüber der naiven Suche zu erwarten ist. Sind jedoch nur wenige Pools relevant, so werden die Kosten deutlich geringer. Bei dem *caching*-basierten Verfahren ist zu beachten, dass die Synchronisationskosten lediglich bei dem *initialen* Austausch von Prototypen mit quadratischem Aufwand verbunden sind. Spätere Aktualisierungen können hingegen mit linearen Aufwand durchgeführt werden.

Anwendungsbeispiel zur Kostenanalyse			
Kosten bei	naiver Suche	repräsentanten-basierter Suche	caching-basierter Suche
$\|PZ\| = 1$	$K_{\text{naiv}}^{\text{relevant}} = 30.000$	$K_{\text{repr}}^{\text{relevant}} = 2.300$	$K_{\text{sync}}^{\text{relevant}} = 2.400$ $K_{\text{cache}}^{\text{relevant}} = 2.000$
$\|PZ\| = 15$	$K_{\text{naiv}}^{\text{relevant}} = 30.000$	$K_{\text{repr}}^{\text{relevant}} = 30.300$	$K_{\text{sync}}^{\text{relevant}} = 2.400$ $K_{\text{cache}}^{\text{relevant}} = 30.000$

Tabelle 5.7.2: Konkretes Anwendungsszenario zur Kostenanalyse

5.7.2 Identifikation von Gemeinschaften und Repräsentanten

Wie eingangs erläutert soll in diesem Abschnitt auf die Identifikation von Gemeinschaften in Netzgraphen sowie der Bestimmung von Repräsentanten in diesen eingegangen werden. Das Ziel ist dabei einen Vergleich zwischen unterschiedlichen Ansätzen und Verfahren zu ziehen, die hierfür geeignet sind.

Gemeinschaften Zur Erkennung von Gemeinschaften bzw. Pools in Netzwerken, wie sie im Rahmen der Problemstellung (Abbildung 1.1.2) vorgestellt wurden, sind die folgenden Verfahren einsetzbar.

- **Affinity Propagation** Dieses Clustering-Verfahren eignet sich für gewichtete Netzgraphen und ist in der Lage, sowohl Gemeinschaften (Cluster), als auch Repräsentanten (Exemplare) zu identifizieren. Zum besseren Verständnis wurden die für den Beispielgraph aus Abbildung 2.7.3 identifizierten Gemeinschaften und Repräsentanten in der folgenden Abbildung 5.7.1 zusammengefasst dargestellt.

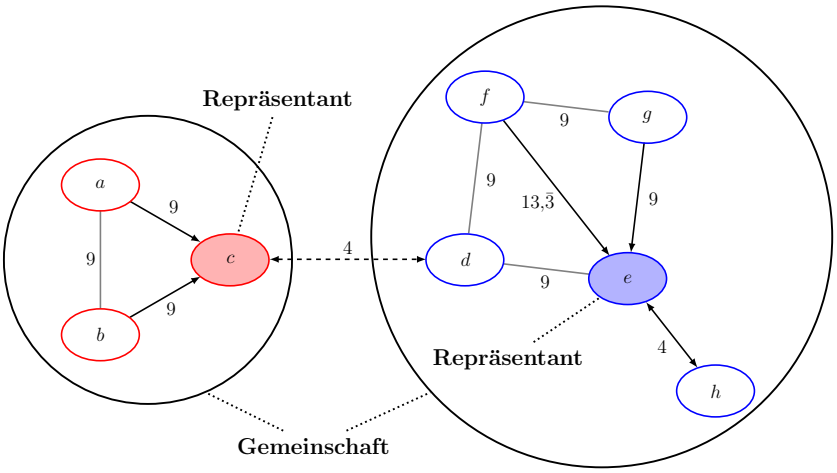


Abbildung 5.7.1: AfP zur Erkennung von Gemeinschaften und Prototypen

- **Ant Colony Optimization** Liegen, im Gegensatz zum vorherigen Verfahren, ungewichtete Netzgraphen vor, so ist AfP nicht nutzbar. In einer solchen Situation kann die ACO eingesetzt werden, um Gemeinschaften zu identifizieren. Ausgehend von der Abbildung 2.7.6, die die erfolgreiche Zerteilung des ungewichteten Beispielgraphen zeigt, wurden die gefundenen Gemeinschaften in der Abbildung 5.7.2 visualisiert.

Repräsentanten Wurden Gemeinschaften identifiziert, so müssen in diesen Repräsentanten bestimmt werden. Dabei können – je nach Art des vorliegenden Netzgraphen – unterschiedliche Kriterien zur Bestimmung genutzt werden.

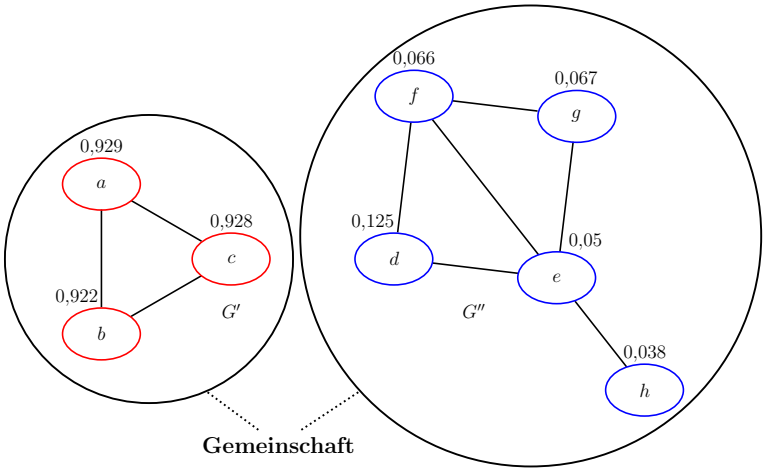


Abbildung 5.7.2: ACO zur Erkennung von Gemeinschaften

- **Knotengrad** Dieser Ansatz ist besonders für ungewichtete Graphen geeignet, in denen lediglich der Knotengrad bestimmbar ist. Der Grad eines Knotens r kann, wie in Gleichung (2.2.1) definiert, durch

$$\deg(r) := \sum_{j=1}^{n} a_{rj}$$

berechnet werden. Da sich ein geeigneter Repräsentant dadurch auszeichnet, dass er über einen hohen Vernetzungsgrad verfügt, kommen somit alle Knoten r für einen optimalen Repräsentanten r^* infrage, für die

$$\deg(r^*) = \max(\deg(r)) \; \forall r, r^* \in P$$

gilt.

- **Kantengewichte** Liegen Gewichtungen in Form von beispielsweise Übertragungskosten vor oder wurden zuvor berechnet (siehe Abschnitt 5.5.2), so kann eine weiteres Kriterium für die Wahl eines Repräsentanten in den minimalen Übertragungskosten zu allen übrigen Knoten liegen. Dies kann durch Bestimmung eines optimalen r^* mittels kostenminimaler Pfade $\mathcal{W}_{r,p}^{\min}$ (z.B. Dijkstra) geschehen:

$$r^* \in P : K(r^*) \rightarrow \min!$$
$$\text{mit } K(r) = \sum_{p \in P \setminus \{r\}} \sum_{(i,j) \in \mathcal{W}_{r,p}^{\min}} k(i,j)$$

- **Affinity Propagation** Der dritte Ansatz ist die Verwendung von AfP. Hierbei entstehen durch das Clustering, wie bereits in Abschnitt 4.3.1

evaluiert, Exemplare, die als Repräsentanten betrachtet werden können. Siehe hierzu Abbildung 5.7.1.

Der Vorteil einer Bestimmung von Repräsentanten anhand von *Knotengraden* liegt darin, dass der Knoten mit maximalem Grad leicht bestimmbar ist. Jedoch ist diese Methode nicht unbedingt sehr aussagekräftig, da die alleinige Anzahl an Kanten die mit einem Knoten inzidieren die Gewichtungen nicht berücksichtigen.

Geeigneter ist (sofern beispielsweise Übertragungskosten vorliegen) eine Bewertung von *Kantengewichten*. Hierbei entsteht zwar ein höherer Aufwand in der Berechnung des optimalen r^*, jedoch lassen sich präzisere Aussagen über einen Knoten treffen, ob er als Repräsentant infrage kommt, da seine Latenzen zu den übrigen Knoten berücksichtigt werden.

Das von den untersuchten Verfahren am besten geeignetste ist allerdings die *Affinity Propagation*, da Kantengewichte berücksichtigt werden und bei dieser Methode keine ergänzendes Verfahren notwendig ist.

6 Zusammenfassung und Fazit

Unternehmen, die zur Unterstützung bei einer kollaborativen Produktentwicklung auf den Einsatz von PDM-Systemen zur Verwaltung ihrer Produktmodelle zurückgreifen, stehen bei einem zentralisierten Ansatz vor nicht zu unterschätzenden Problemen. Gerade bei einer großen Anzahl externer Kollaborationspartner ist eine effiziente Datenverwaltung mit einer zentralen PDM-Instanz aufgrund der „*Flaschenhalsproblematik*“ nicht sinnvoll. Hier bieten sich eher dezentrale Ansätze an, um Ressourcen zwischen mehreren Beteiligten effizient austauschen zu können. Insbesondere strukturierte P2P-Systeme, in denen Ressourcen mittels eindeutigen Hash-Werten identifiziert werden, erlauben eine effiziente Lokalisierung von Produktdaten, sofern deren Bezeichner (ID) genau bekannt ist. Da jedoch die alleinige Kenntnis einer ID nicht ausreicht, um auf den Inhalt der Ressource schließen zu können und oftmals der Inhalt der zu suchenden Ressource nicht präzise bekannt ist, ist dieser Ansatz bei einer Suche nach **ähnlichen Daten** nicht zielführend.

Hierbei stellte sich im Rahmen dieser Arbeit zentral die Frage, wie eine solche Lokalisierung erfolgen kann, ohne naiv sämtliche Ressourcen abgleichen zu müssen. Gerade in Netzwerken mit vielen Teilnehmern und Produktmodellen würde so ein signifikant hoher Aufwand entstehen. Handelt es sich zudem um Netzwerke, die aus einzelnen, spärlich vernetzten Pools bestehen, so können zudem die Übertragungskosten zwischen den Pools deutlich zu dem Gesamtaufwand beitragen.

Zur Lösung dieser Probleme war es daher notwendig Strategien zu entwerfen, wie der zu betrachtende Suchraum frühzeitig minimiert werden kann (beispielsweise durch Ausschluss nicht relevanter Pools), um kostspielige Abgleiche einsparen zu können.

Im Rahmen dieser Arbeit wurden daher unterschiedliche Ansätze entworfen und vorgestellt, die eine inhaltsbasierte Lokalisierung ähnlicher Produktmodelle in dezentralen Kollaborationsumgebungen zum Ziel haben.

Es wurden zu diesem Zweck bereits bestehende Verfahren bzw. Systeme wie *OceanStore*, *Kademlia* sowie *Squid* in Kapitel 3 näher untersucht, jedoch war keines von diesen in der Lage, die Anforderungen an eine effiziente inhaltsbasierte Suche von Ressourcen in verteilten Umgebungen zu erfüllen.

Daher wurden in Kapitel 4 die notwendigen Anforderungen an eine solche Lokalisierung auf der Grundlage von strukturierten P2P-Overlay-Netzwerken verfeinert und Modelle entworfen, die das Problem der ähnlichkeitsbasierten Suche in verteilten Systemen lösen sollen. Um eine Unabhängigkeit von konkreten P2P-Implementierungen zu behalten, wurden die Modelle so konzipiert,

dass sie als Erweiterung auf beliebige P2P-Backends aufsetzen können. Für die Ermittlung von ähnlichen Daten boten sich Clustering-Verfahren an, die in der Lage waren Prototypen von Clustern zu bestimmen. Diese Prototypen dienten dann im weiteren Verlauf als Referenz zu Identifikation relevanter Cluster, die gesuchte Ressourcen enthalten. Bei der Suche nach Clustering-Verfahren zur Bestimmung von Prototypen zeigte sich, dass, bei vorliegenden Ähnlichkeiten bzw. Distanzen zwischen Daten, *k-means* (sowie Verbesserungen hiervon) und *Affinity Propagation (AfP)* am besten geeignet sind. Affinity Propagation ist zudem auch in der Lage auf Graphenstrukturen zu operieren.

Im Abschnitt 4.2 wurde auf Basis dieser Verfahren das *Clustering Data Model (CLDM)* spezifiziert. Dieser Ansatz löste zwar das Problem der Lokalisation von ähnlichen Daten, zeigte jedoch eine signifikante Schwäche in Form der zentralen Cluster-Instanz, die dem fundamentalen Prinzip von Peer-to-Peer-Systemen widerspricht. Ein deutlich besserer Ansatz zeigte sich im Abschnitt 4.2.7 in Form vom *Locality-Sensitive Data Model (LSDM)*, welches auf den Prinzipien lokal sensitiver Hash-Funktionen basiert und eine dezentral organisierte Verwaltung von ähnlichen Ressourcen in P2P-Netzwerken bietet. Hierbei werden von LS-Peers, die als Super-Peers in dem Netzwerk fungieren, Informationen in sogenannten *Buckets* darüber bereitgestellt, welche Ressourcen ähnlich zueinander sind. Durch Anwendung von LSH-Funktionen auf Referenzspezifikationen, die als Suchkriterium dienen, kann ein gewöhnlicher Peer leicht berechnen, welcher LS-Peer für ein Bucket verantwortlich ist und so die ähnlichen Ressourcen identifizieren und alle übrigen als nicht relevant ausschließen.

Im weiteren Verlauf wurde gezeigt, dass das LSDM gegenüber der CLDM deutliche Vorteile besitzt, da es über keine zentrale Instanz verfügt und darüber hinaus durch den Einsatz der approximativen LSH-Funktionalität (mit einem vertretbaren Maß an Ungenauigkeit) effizient alle ähnlichen Ressourcen in linearer Zeit identifizieren kann.

In dem zweiten Teil dieser Arbeit (Kapitel 5) wurde eine ähnlichkeitsbasierte Verwaltung hinsichtlich weit verteilter, sich in Pools organisierender, Netzwerke entworfen. Hierbei spielte die zuvor beschriebene Problematik, dass die Lokalisierung von ähnlichen Ressourcen, die gerade wegen der großen Distanzen zwischen den Pools sehr kostenintensiv sein kann, eine zentrale Rolle.

Zur Lösung des Problems wurden drei mögliche Suchstrategien aus der Perspektive eines Teilnehmers vorgestellt, die der Lokalisierung von ähnlichen Ressourcen dienen. Der erste und zudem naive Ansatz sah vor, dass ein Teilnehmer, ausgehend von einer Referenzspezifikation als Suchkriterium, alle Ressourcen im Netzwerk abfragt und mit seiner Referenz abgleicht. Diese Strategie

ist zwar langfristig erfolgreich, jedoch mit einem enormen Aufwand durch die hohen Inter-Pool-Kosten verbunden. Als eine signifikante Verbesserung zeigte sich daher die Strategie, dass in jedem Pool ein Repräsentant einen Prototypen der dort verwalteten Daten bereitstellt. Somit muss ein Teilnehmer der nach Datenobjekten sucht, die ähnlich zu seinen Kriterien sind, nur noch eine Anfrage an jeden Pool übertragen und kann vor seiner eigentlichen Suche bereits nicht relevante Pools ausschließen. Eine Weiterentwicklung von diesem Vorgehen stellte die dritte Strategie dar. Sie löste das Problem, dass bei jeder Suchanfrage sämtliche Prototypen aus entfernten Pools abgefragt werden müssen, indem die Repräsentanten alle Prototypen austauschen und mittels *caching* vorhalten.

Abschließend wurden Verfahren vorgestellt wie Gemeinschaften (Pools) und Repräsentanten in Netzwerken identifiziert werden können. Hierbei zeigte sich *Affinity Propagation* als sehr vielversprechend (sofern das Netzwerk in einen gewichteten Graphen überführbar ist), da es sowohl Gemeinschaften identifizieren kann, als auch Repräsentanten bestimmt. Liegen jedoch ungewichtete Graphen vor, so kann AfP nicht eingesetzt werden. Statt dessen zeigte sich hier der Einsatz der *Ant Colony Optimization (ACO)* zur Erkennung von Gemeinschaften als nützlich. Diese kann zudem auch verteilt mittels Agenten durchgeführt werden und wurde im Rahmen dieser Arbeit für die Multiagentenplattform *SPADE* implementiert. Da durch die ACO jedoch keine Prototypen identifizierbar sind, bietet sich bei ungewichteten Graphen lediglich eine Erkennung über Knotengrade an.

Abschließend lässt sich sagen, dass sich eine Lokalisierung von Produktmodellen in dezentralen Umgebungen durch inhaltsbasierte Verteilungsstrategien mit den in dieser Arbeit untersuchten und vorgestellten Lösungskonzepten effizient durchführen lässt.

7 Ausblick

In diesem Abschnitt werden Aspekte erläutert, die sich als sinnvolle Ziele weiterführender Arbeiten anbieten. Hierbei behandelt ein Aspekt die Qualität ähnlichkeitsbasierter Gruppierungen in Netzwerken mit hohen Fluktuationen hinsichtlich verfügbarer Daten. Ein weiterer Punkt liegt in der sicherheitskritischen Betrachtung von Suchvorgängen, wenn sich vertrauliche Daten in nichtlokalen Pools befinden. Des Weiteren wird auf mögliche Erweiterungen der im Rahmen dieser Arbeit vorgestellten Datenmodelle eingegangen.

7.1 Sicherheitskritische Verwaltung von Ressourcen in entfernten Pools

Wie in Abschnitt 5.2 erläutert wurde, sollen bei der dezentralen Ressourcenverwaltung in Netzwerkpools Daten so abgelegt werden, dass alle Daten die ähnlich zueinander sind, in einem Pool zu finden sind. Gerade in nicht vertrauenswürdigen P2P-Netzwerken entstehen bei diesem Ansatz Risiken, da die Ressourcen prinzipiell von jedem Teilnehmer, dem sie zugeordnet werden, gelesen werden können. In dem Kontext der kollaborativen Produktentwicklung zwischen mehreren Unternehmen ist zwar denkbar, dass externe Anfragen in eigene Pools kontrolliert und ggf. abgelehnt werden können, jedoch können aufgrund der ähnlichkeitsbasierten Gruppierung auch Unternehmensinterne Daten in fremde Pools gelangen, sofern diese mit verwaltet werden sollen. Eine allgemeine Lösung wäre die Verschlüsselung der Daten bevor sie an fremde Pools übergeben werden. Jedoch können durch diesen Ansatz nur sehr ineffizient Such- und Vergleichsanfragen über große Distanzen durchgeführt werden (siehe Abschnitt 5.3.3, `proxy_compare()`), da Ressourcen zunächst vollständig übertragen werden, lokal entschlüsselt und anschließend verglichen werden müssten. In den Arbeiten von Halloush und Sharif [HS09] sowie Song, Wagner und Perrig [SWP00] werden die Probleme, wie chiffrierte Daten auf entfernten Servern gesucht und verglichen werden können aufgegriffen und diesbezüglich interessante Lösungsansätze vorgestellt. Schwerpunkte künftiger Arbeiten könnten daher in einer sicherheitsorientierten Betrachtung der verteilten Suche in P2P-basierten Produktentwicklungsumgebungen liegen, mit der Fragestellung, ob auf *Gatekeeper* (siehe Abschnitt 5.3.3) durch einen solchen Ansatz verzichtet werden könnte.

7.2 Gewichtung einzelner Merkmalsattribute im Rahmen der LSDM

Ein weiterer interessanter Aspekt des in Abschnitt 4.2.7 vorgestellten Locality-Sensitive Data Model stellt die Gewichtung unterschiedlicher Ausprägungen von Daten-Attributen dar. Lassen sich bestimmte Eigenschaften von Ressourcen in relevante und weniger relevante klassifizieren, so können ganz allgemein Datenobjekte die aufgrund eines *euklidischen Abstands* unähnlich sind, durch stärkere Gewichtung der relevanten Attribute, als ähnlich identifiziert und entsprechend gruppiert werden. Dieser Aspekt könnte für Unternehmen, die sich nur für bestimmte Attribute von Baugruppen bzw. Bauteilen interessieren und andere als weniger relevant erachten, sinnvoll sein.

Daher sollte das Ziel weiterführender Untersuchungen sein, auf welche Weise Produkte (die sich nur in einer oder wenigen konkreten Ausprägungen gleichen) gruppiert werden können, ob ein Mehrwert für branchenübergreifende Kollaborationen entsteht und darüber hinaus inwiefern dieses Konzept mittels LSH umgesetzt und im Rahmen der LSDM eingesetzt werden kann.

7.3 Qualität der Cluster bei hohen Datenfluktuationen

In P2P-Systemen kann im Allgemeinen davon ausgegangen werden, dass sich die bereitgestellten Daten häufig ändern, neue hinzukommen oder bestehende entfernt werden. Hierbei stellt sich die Frage, wie groß der Aufwand zur Erhaltung einer qualitativ hochwertigen Datengruppierung sein muss, gerade im Hinblick auf hohe Datenfluktuationen.

Das **Clustering Data Model** (Abschnitt 4.2.4) setzt ein Clustering ein, bei dem die Prototypen direkt von den Daten innerhalb der Cluster abhängen. Somit muss die Cluster-Instanz bei jeder Änderung des Datenbestandes die von ihr verwalteten Cluster durch zusätzlichen Rechenaufwand aktualisieren, damit die Prototypen im Schwerpunkt der Cluster verbleiben. In diesem Zusammenhang stellt sich für eine weitergehende Untersuchung die Frage, ob ein erneutes Clustern der Daten stets erforderlich ist oder in bestimmten Situationen entfallen kann, ohne signifikante Einbußen in der Qualität der Cluster zu erleiden. Gerade bei dem Einsatz von Verfahren wie Affinity Propagation sollte untersucht werden, ob bei einem erneuten Nachrichtenaustausch (aufgrund von modifizierten Daten) lediglich die veränderten Bereiche des Nachbarschaftsgraphen betroffen sind und wie sich lokale Änderungen von Datenpunkten auf den gesamten Graphen auswirken bis die *availability*- und *responsibility*-Werte erneut konvergieren.

Bei dem **Locality-Sensitive Data Model** (Abschnitt 4.2.7) entsteht der hauptsächliche Aufwand hingegen bei den Teilnehmern, die Ressourcen veröffentlichen. Die Qualität der Zuordnungen ist davon allerdings unberührt, da das Clustering ausschließlich über LSH-Funktionen geschieht, die beim Hinzufügen von Ressourcen bereits erzeugt wurden. Allerdings muss bei Überschreitung von maximal zulässigen Attributsgrenzen ein *rehashing* aller Daten durchgeführt werden. Dieser Vorgang kann durchaus Auswirkungen auf die Qualität der Nachbarschaften haben, da es zu einer Vergrößerung des LSH-Vektorraums kommt. Da der LSH-Vektorraum in dem bisherigen Design der LSDM nur vergrößert werden kann, um Vektoren mit größeren Attributwerten zuzulassen, kann es bei der Bestimmung von Nachbarschaften zu Problemen kommen, wenn durch das Entfernen von Daten viele leere Bereiche entstehen. Es ist daher für weitere Arbeiten lohnenswert, sich mit einer dynamischen Skalierbarkeit des LSH-Vektorraums zu beschäftigen, um negative Effekte bei der Bestimmung von Nachbarschaften zu vermeiden.

Anhang

Algorithmus A.1 Breadth-First Search [Jun07, S. 64]

Input: Adjacency lists A_v of Graph G with uniform weighted edges, $s \in V(G)$, queue Q, vertices v of G labeled with $d : v \to \mathbb{N}$

procedure BFS(G, s, d)
  $Q \leftarrow \emptyset$; $d(s) \leftarrow \emptyset$
  append s to Q
  while $Q \neq \emptyset$ **do**
    remove first vertex v from Q
    for $w \in A$ **do**
      if $d(w)$ is undefined **then**
        $d(w) \leftarrow d(v) + 1$
        append w to Q

Output: Distances with respect to vertex s,

$$d(s,t) = \begin{cases} d(t) & \text{if } d(t) \text{ is defined} \\ \infty & \text{otherwise} \end{cases} \quad \forall t \in V(G)$$

Algorithmus A.2 Dijkstra [Jun07, S. 76f]

Input: Network (G, w) with $w(e) \geq 0 \; \forall e \in V(G)$, adjacency lists A_v, vertices v of G labeled with $d : v \to \mathbb{N}$

procedure DIJKSTRA(G, w, s, d)
  $d(s) \leftarrow 0$
  $T \leftarrow V(G)$
  for $v \in V(G) \setminus \{s\}$ **do**
    $d(v) \leftarrow \infty$
  while $T \neq \emptyset$ **do**
    find some $u \in T$ such that $d(u)$ is minimal
    $T \leftarrow T \setminus \{u\}$
    for $v \in T \cap A_u$ **do**
      $d(v) \leftarrow \min\{d(v), d(u) + w_{uv}\}$

Output: Distances with respect to vertex s, $d(s,t) = d(t) \; \forall t \in V(G)$

Algorithmus A.3 Floyd and Warshall [Jun07, S. 84]

Input: Network (G, w) with $w(e) \in \mathbb{R}\ \forall e \in V(G)$ not containing any cycles of negative length, $V(G) = \{1, \ldots . n\}$.
Set $w(e_{ij}) = \infty$ if $e_{ij} \notin V(G)$

```
procedure FLOYD(G, w, d)
    for i = 1 → n do
        for j = 1 → n do
            if i ≠ j then
                d(i, j) ← w(e_ij)
            else
                d(i, j) ← 0
    for k = 1 → n do
        for i = 1 → n do
            for j = 1 → n do
                d(i, j) ← min{d(i, j), d(i, k) + d(k, j)}
```

Output: Distances with respect to vertex s, $d(s,t) = d(t)\ \forall t \in V(G)$

Algorithmus A.4 Common Neighborhood Density [KC09]

Input: Adjacency matrix $A^{n \times n}$, parameter γ

```
procedure K(i, j)
    C_ij ← {i, j}                          ▷ Index set of common neighbors
    K_ij ← 0
    for k = 1 → n do
        if A_ik ≠ 0 and A_jk ≠ 0 then
            C_ij ← C_ij ∪ {k}
    for k in C_ij do
        for l in C_ij do
            K_ij ← K_ij + A_kl
        K_ij ← K_ij · |C_ij|^γ / ( |C_ij| · (|C_ij| − 1) )
    return K_ij

K^{n×n} ← 0                                ▷ Edge density matrix of common neighbors
for i = 1 → n do
    for j = i + 1 → n do
        K_ij ← K(i, j)
```

Output: Edge density matrix $K^{n \times n}$

Algorithmus A.5 ACO: Heuristic [Man+11, S. 9]

Input: Number of ants n_a, maximum number of iterations T, solution s^{kt} constructed by ant k during iteration t, best solution s_{tb} among all ants in iteration t, best solution s_{rb} since last reset and best solution s_{sb} so far.

init $t = 0; \tau_i = 0.5; n_a; T$
while $t < T$ **do**
  for $k = 1 \to n_a$ **do**
    $s^{kt} \leftarrow$ ConstructSolution(τ)
    $s^{kt} \leftarrow$ LocalSearch(τ)
  $\tau \leftarrow$ PheromoneUpdate($\tau, s_{tb}, s_{rb}, s_{sb}$)
  if Convergence **then**
    Reset τ and s_{rb}
  $t \leftarrow t + 1$

Output: Pheromone values τ, solution s_{rb}

Algorithmus A.6 ACO: Local Search [Man+11, S. 10]

Input: Solution $s \in S^*, S^* = \{s_{tb}, s_{rb}, s_{bs}\}$
procedure LocalSearch(s)
  $g_i(s) \leftarrow b_{ii}(1 - 2s_i) + 2\sum_{j=1, j\neq i}^{n} b_{ji} s_j (1 - 2s_i)$
  while $\max_j g_j(s) > 0$ **do**
    $i \leftarrow \arg\max_{j=1,\dots,n}\{g_j(s)\}$
    $s_i \leftarrow 1 - s_i$
    $g_j(s^{\text{new},i}) \leftarrow \begin{cases} -g_i(s) & j = i \\ g_j(s) + 2b_{ji}(1 - 2s_j)(2s_i - 1) & j \neq i \end{cases}$ ▷ update gains
  return s

Algorithmus A.7 k-means [Ert08, S. 228]

Input: Elements $x_i \in \mathbb{R}^n, i = 1, \dots, m$, number k of desired clusters
init $w_i, i = 1, \dots, k$ (e.g. random)
repeat
  $w_i \leftarrow \{x_j \,|\, \|x_j - w_i\| < \|x_j - w_l\| \; \forall i \neq l\}$
  $w^i \leftarrow \frac{1}{|w_i|} \sum_{x_j \in w_i} x_j$
until w **converges**

Output: w

Algorithmus A.8 Affinity Propagation (*median preferences*) [FD07b, S. 2f]

Input: Similarity matrix $S = (s_{ik})$, iterations n, dampening parameter λ

$s_{jj} \leftarrow \text{median}_{i,k:i\neq k}\, s_{ik}$ ▷ set median preferences

$N \leftarrow \text{length}(S, 0)$ ▷ number of columns or rows

init matrices A, R with $a_{ik} = r_{ik} = 0\ \forall i, k = 1, \ldots, N$

for $i = 1 \rightarrow n$ **do**

$r_{ik}^{\text{old}} \leftarrow r_{ik}$

$a_{ik}^{\text{old}} \leftarrow a_{ik}$

$r_{ik} \leftarrow s_{ik} - \max_{k' \text{ s.t. } k'\neq k} \{a_{ik'} + s_{ik'}\}$

$a_{ik} \leftarrow \min \left\{0, r_{kk} + \sum_{i' \text{ s.t. } i'\notin\{i,k\}} \max\{0, r_{i'k}\}\right\}$ für $i \neq k$

$a_{kk} \leftarrow \sum_{i' \text{ s.t. } i'\neq k} \max\{0, r_{i'k}\}$

$r_{ik} \leftarrow (1-\lambda)\cdot r_{ik} + \lambda \cdot r_{ik}^{\text{old}}$ ▷ dampen responsibilities

$a_{ik} \leftarrow (1-\lambda)\cdot a_{ik} + \lambda \cdot a_{ik}^{\text{old}}$ ▷ dampen availbilities

$E \leftarrow R + A$ ▷ evidence

$I \leftarrow \{e_{jj} \mid e_{jj} > 0\}$ ▷ indices of exemplars

$K \leftarrow |I|$

$S^I \leftarrow (s_{ik})\ \forall k \in I$ ▷ k-th columns of S

$c \leftarrow (\text{argmax}(S^I))$ ▷ indices of largest value in each row

$c(I) \leftarrow (0, \ldots, K-1)$ ▷ cluster assignments

$idx \leftarrow \{I_i \mid i \in c\}$ ▷ exemplar assignments

Output: c,idx,I

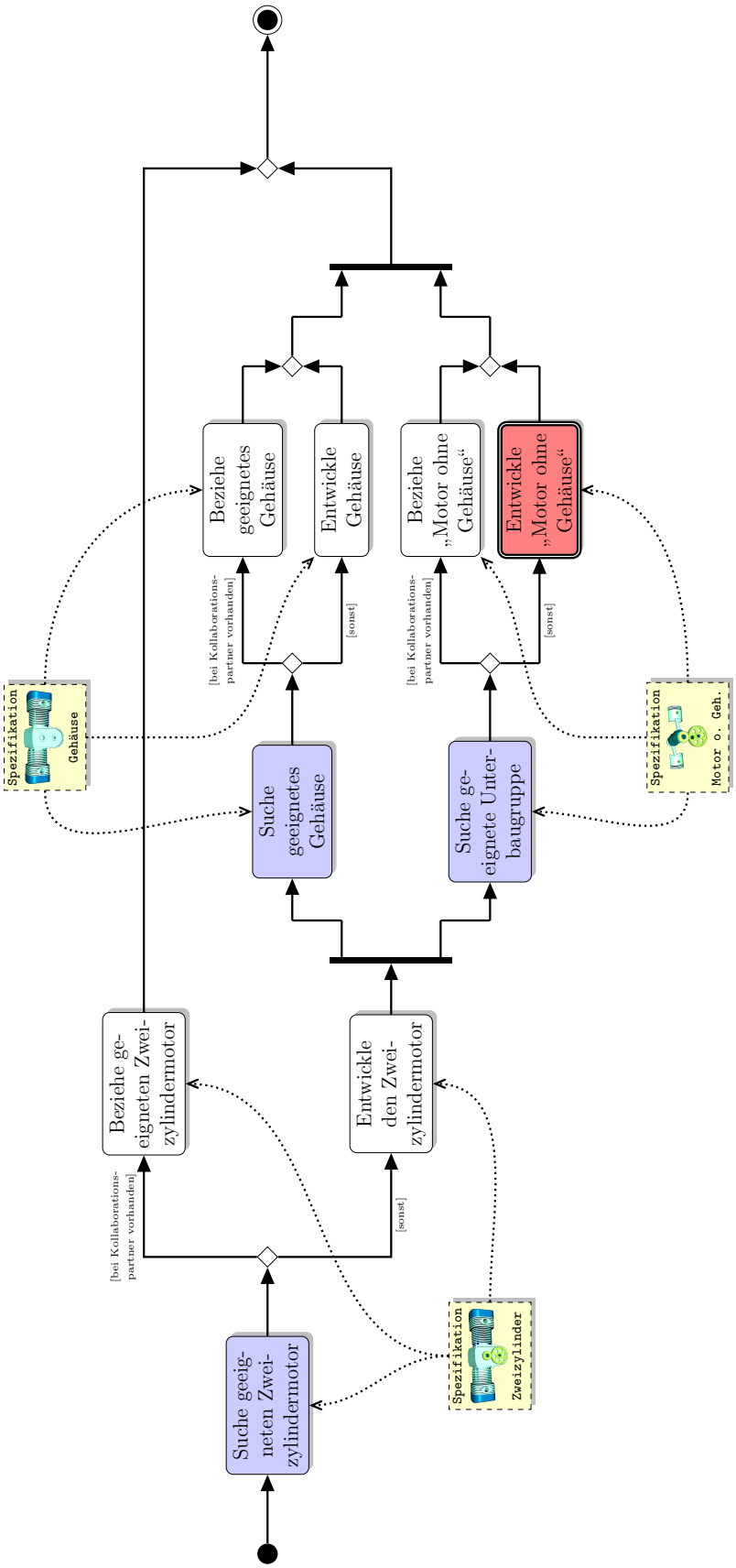


Abbildung A.1: Ablauf der kollaborativen Produktentwicklung mittels inhaltsbasierter Suche (Baugruppen aus [Sie12a])

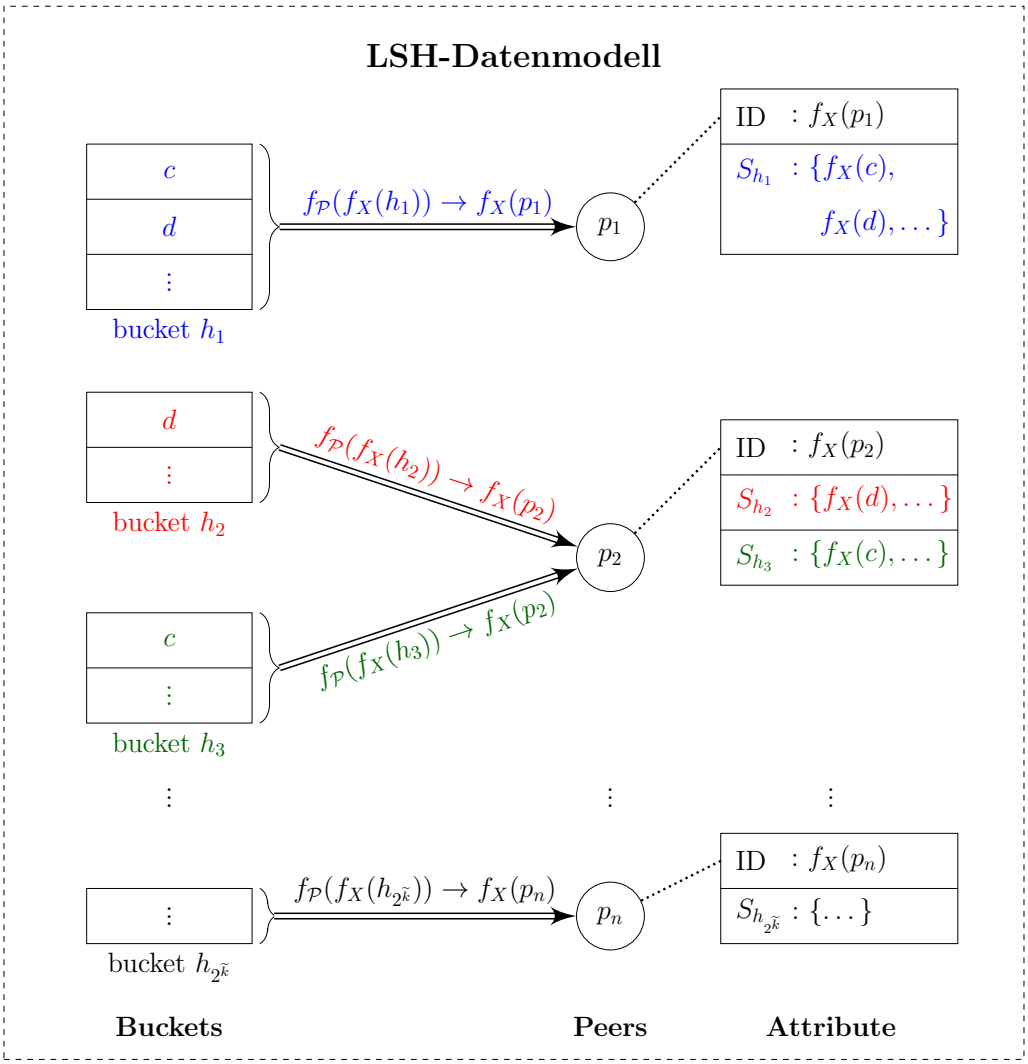


Abbildung A.2: Erweiterung des LSH-Verfahrens

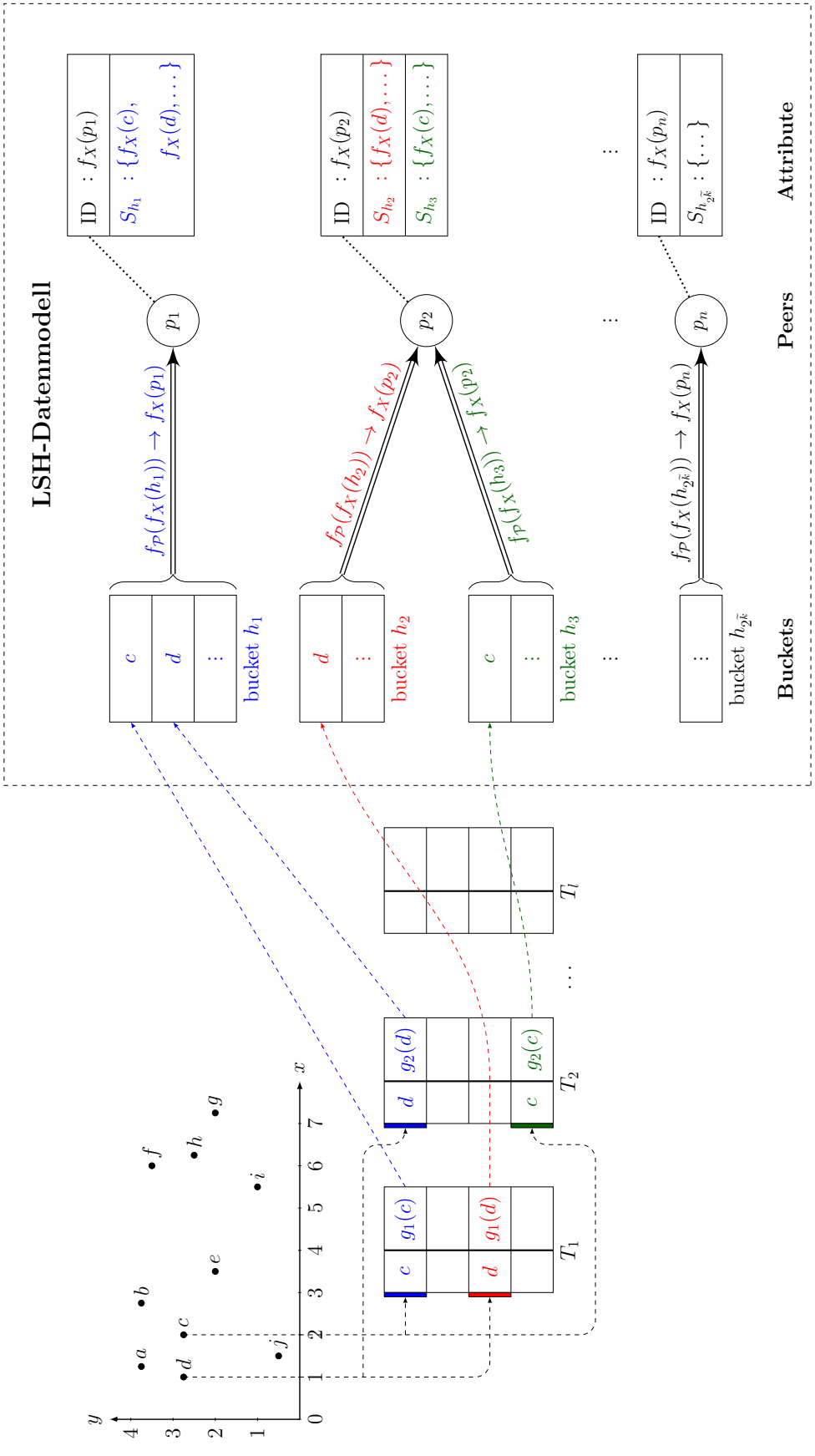


Abbildung A.3: LSH-basiertes Datenmodell

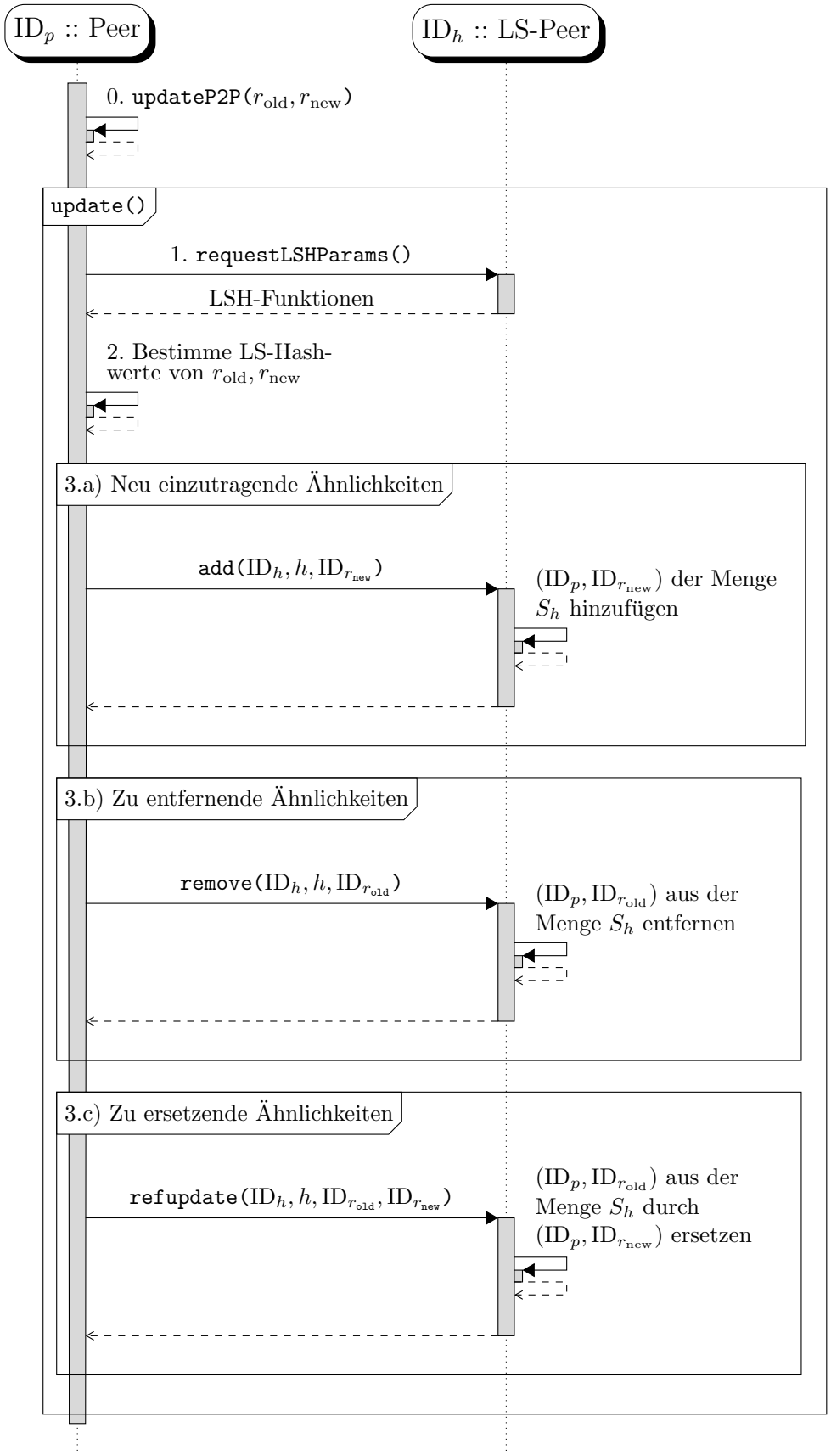


Abbildung A.4: Ablauf der `update()`-Operation im LSDM

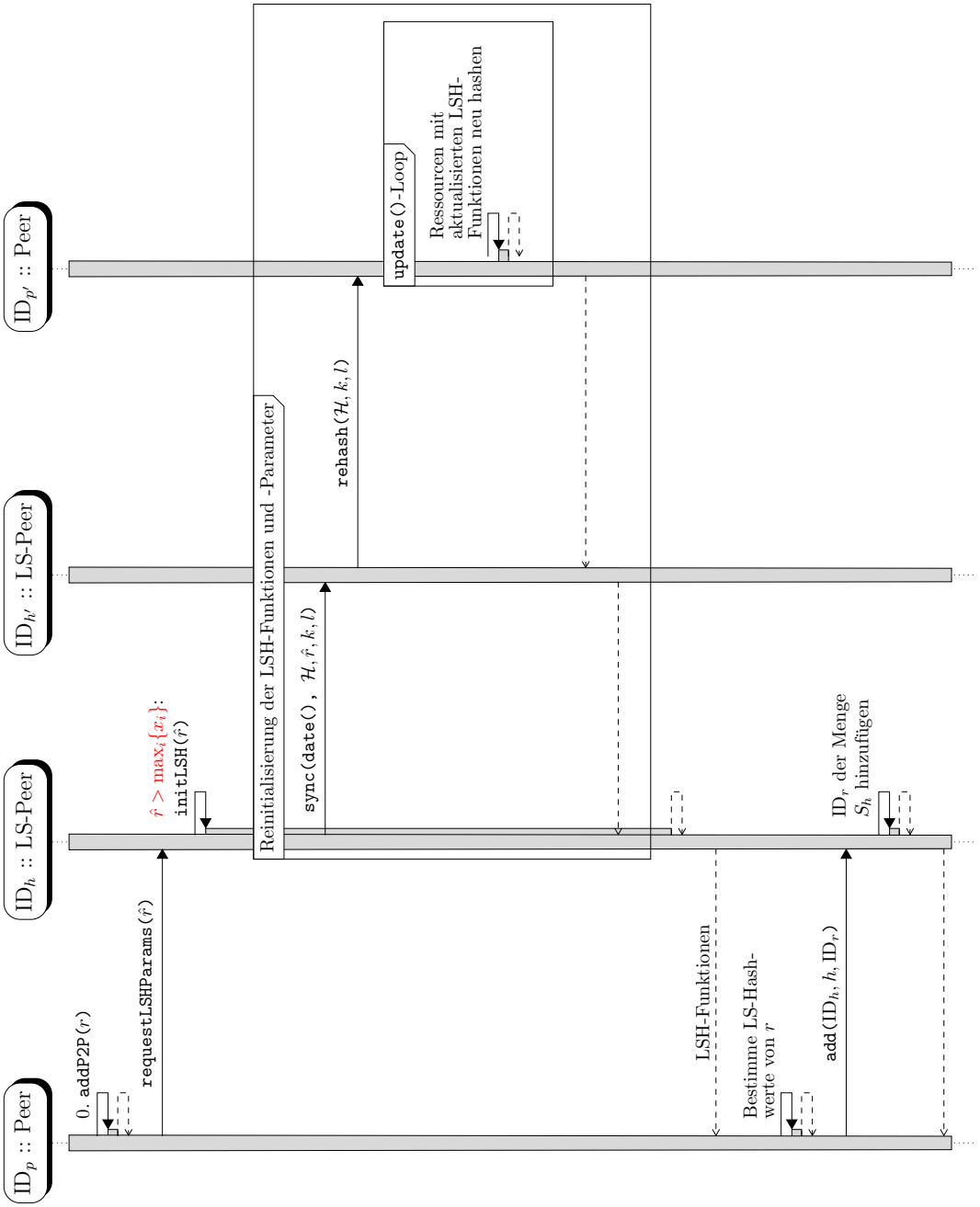


Abbildung A.5: Ablauf der `add()`-Operation im LSDM bei Überschreitung des maximal zulässigen Vektorattributes

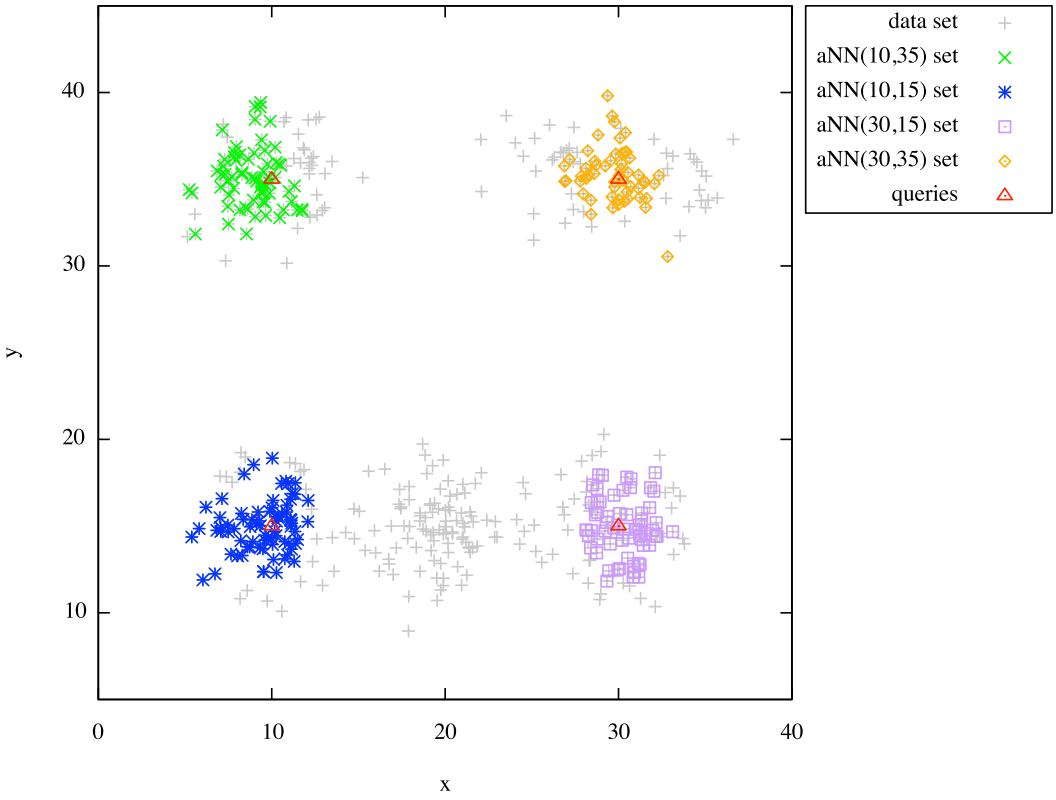
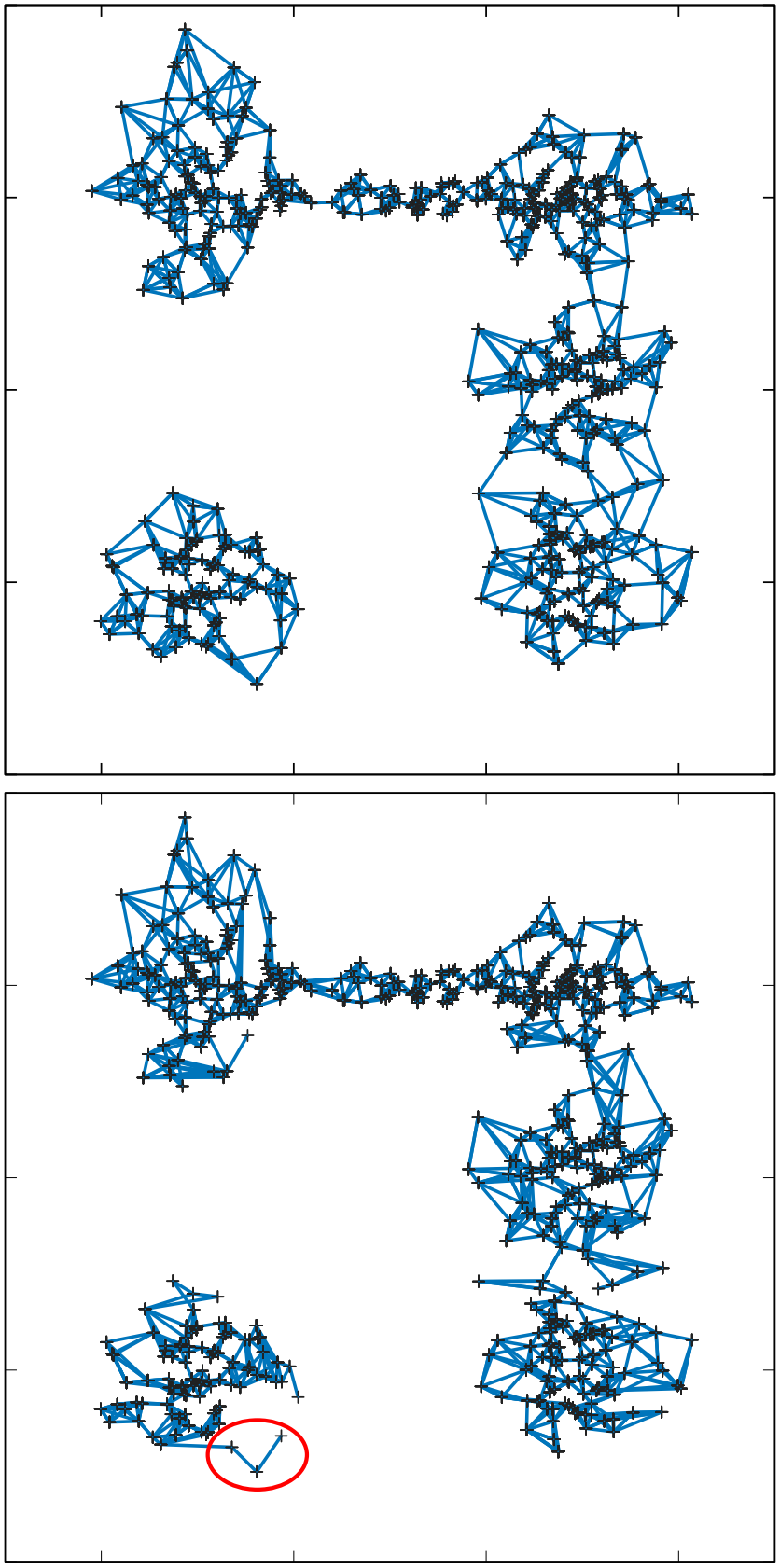


Abbildung A.6: LSH angewendet auf Datenpunkte



(a) Approximatives kNN mittels LSH

(b) Exaktes kNN

Abbildung A.7: Vergleich zwischen exaktem und approximativem kNN

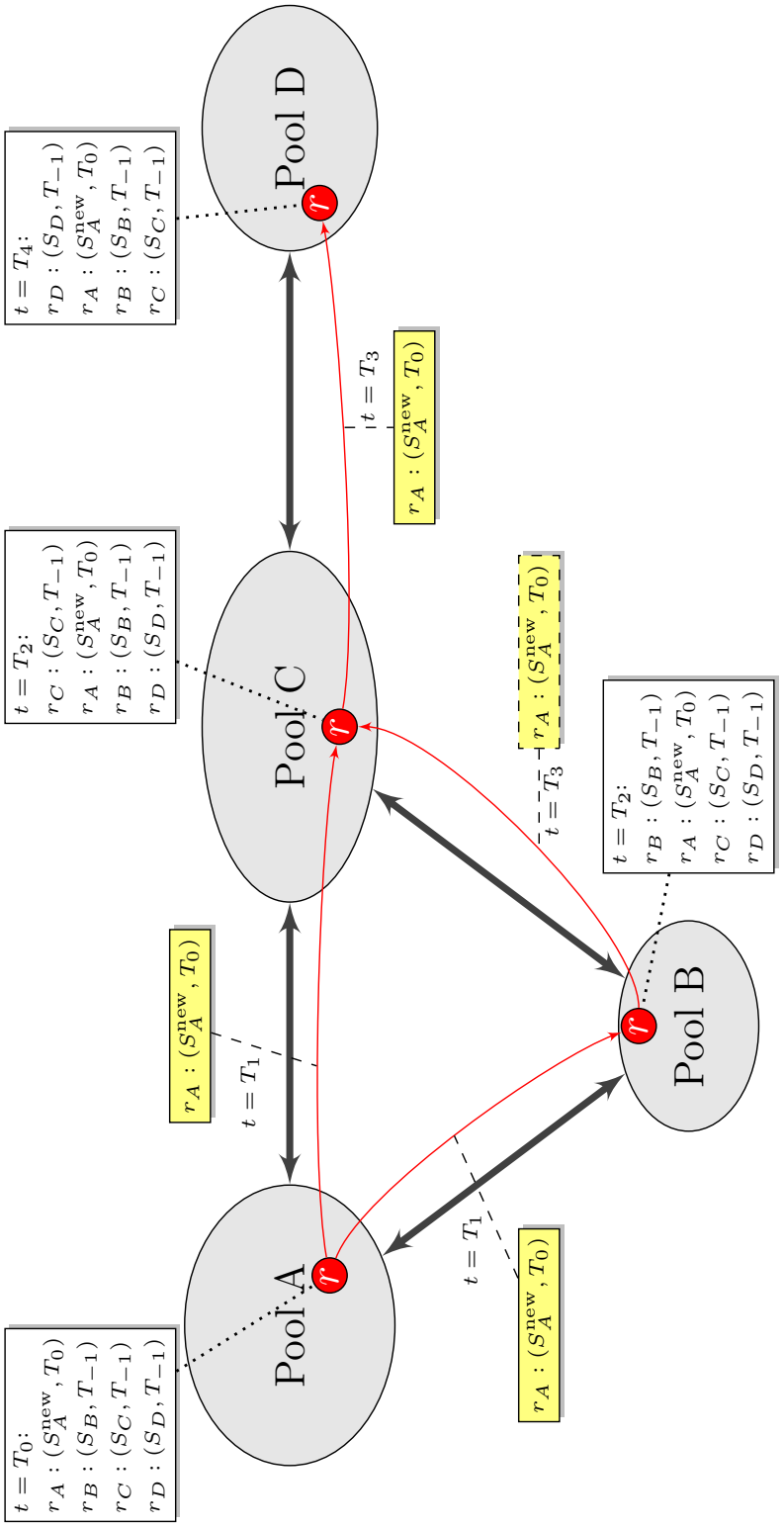


Abbildung A.8: Repräsentant r_A repliziert zu T_0 seine Prototypenspezifikation S_A im Netzwerk mittels `sync()`

Literatur

[Dru+12] Druschel, P. u. a. *FreePastry*. Juli 2012. URL: http://freepastry.org/FreePastry/.

[Ert08] Ertel, W. „Maschinelles Lernen und Data Mining“. In: *Grundkurs Künstliche Intelligenz*. Vieweg+Teubner, 2008, S. 179–240.

[FD07a] Frey, B. J. und Dueck, D. „Clustering by passing messages between data points“. In: *Science* 315 (2007).

[FD07b] Frey, B. J. und Dueck, D. „Supporting Online Material for Clustering by Passing Messages Between Data Points“. In: Science Express, 2007. URL: http://www.sciencemag.org/content/315/5814/972/suppl/DC1.

[For11] Forster, O. *Analysis 2: Differentialrechnung im $\mathbb{R}^n$, gewöhnliche Differentialgleichungen*. 9. Aufl. Grundkurs Mathematik Bd. 2. Vieweg+teubner Verlag, 2011.

[HH07] Hammer, B. und Hasenfuss, A. „Relational Neural Gas“. In: *Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*. KI '07. Osnabrück, Germany: Springer Berlin / Heidelberg, 2007, S. 190–204.

[HS09] Halloush, M. und Sharif, M. „Global Heuristic Search on Encrypted Data (GHSED)“. In: *CoRR* abs/0909.2366 (2009).

[Jun07] Jungnickel, D. *Graphs, Networks and Algorithms*. 3rd. Springer Publishing Company, Incorporated, 2007.

[KC09] Kang, Y. und Choi, S. „Common Neighborhood Sub-graph Density as a Similarity Measure for Community Detection“. In: *Proceedings of the 16th International Conference on Neural Information Processing: Part I*. ICONIP '09. Bangkok, Thailand: Springer-Verlag, 2009, S. 175–184.

[KIW04] Koga, H., Ishibashi, T. und Watanabe, T. „Fast Hierarchical Clustering Algorithm Using Locality-Sensitive Hashing“. In: *Discovery Science*. Bd. 3245. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, S. 155–182.

[KLR04] Keogh, E., Lonardi, S. und Rtanamahatana, C. A. „Toward Parameter-Free Data Mining“. In: *10th ACM SIGKDD*. Seattle, USA, Aug. 2004, S. 206–215.

[Kub+00] Kubiatowicz, J. u. a. „OceanStore: An Architecture for Global-Scale Persistent Storage“. In: *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000))* (2000).

[KWZ98] Krahl, D., Windheuser, U. und Zick, F.-K. *Data Mining – Einsatz in der Praxis*. Addison Wesley, 1998.

[Li+01] Li, M. u. a. „The similarity metric“. In: *CoRR* (2001).

[Lod+02] Lodhi, H. u. a. „Text classification using string kernels“. In: *J. Mach. Learn. Res.* 2 (März 2002), S. 419–444.

[Lux07] Luxburg, U. von. „A Tutorial on Spectral Clustering“. In: *CoRR* abs/0711.0189 (2007).

[Mac67] MacQueen, J. „Some methods for classification and analysis of multivariate observations“. In: *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.* Bd. 1. Univ. of Calif. Press, 1967, S. 281–297.

[Man+11] Mandala, S. u. a. „Clustering social networks using ant colony optimization“. In: *Operational Research* (2011), S. 1–19.

[Mer90] Merkle, R. „One Way Hash Functions and DES“. In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Hrsg. von Brassard, G. Bd. 435. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1990, S. 428–446.

[MM02] Maymounkov, P. und Mazières, D. „Kademlia: A Peer-to-Peer Information System Based on the XOR Metric“. In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, S. 53–65.

[Moh97] Mohar, B. „Some Applications of Laplace Eigenvalues of Graphs“. In: *GRAPH SYMMETRY: ALGEBRAIC METHODS AND APPLICATIONS, VOLUME 497 OF NATO ASI SERIES C*. Kluwer, 1997, S. 227–275.

[MVOV97] Menezes, A., Van Oorschot, P. und Vanstone, S. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and Its Applications. Crc Press, 1997.

[PA12a] Palanca, J. und Aranda, G. *SPADE: API und Dokumentation*. Juli 2012. URL: http://www.javierpalanca.com/spade/.

[PA12b] Palanca, J. und Aranda, G. *SPADE: Projektseite*. Juli 2012. URL: http://code.google.com/p/spade2/.

[Pre98] Preneel, B. „Cryptographic Primitives for Information Authentication — State of the Art“. In: *State of the Art in Applied Cryptography*. Bd. 1528. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1998, S. 49–104.

[RD01] Rowstron, A. und Druschel, P. „Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems“. In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. Heidelberg, Germany, Nov. 2001, S. 329–350.

[Run10] Runkler, T. A. *Data Mining: Methoden und Algorithmen intelligenter Datenanalyse*. Vieweg+Teubner, 2010.

[Shu+05] Shu, Y. u. a. „Supporting Multi-dimensional Range Queries in Peer-to-Peer Systems“. In: *FIFTH IEEE INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING*. 2005, 173-180.

[Sie12a] Siemens. *PLM, JT2Go, 2 Cylinder Engine (typical or "shattered"JT file)*. Juli 2012. URL: http://www.plm.automation.siemens.com/en_us/products/teamcenter/lifecycle-visualization/jt2go/downloads/index.shtml.

[Sie12b] Siemens. *PLM, JT2Go, Solid Edge 853 Roadster (monolithic)*. Juli 2012. URL: http://www.plm.automation.siemens.com/en_us/products/teamcenter/lifecycle-visualization/jt2go/downloads/index.shtml.

[SP03] Schmidt, C. und Parashar, M. „Flexible Information Discovery in Decentralized Distributed Systems“. In: *Proceedings of the 12th High Performance Distributed Computing (HPDC)*. 2003, S. 226–235.

[STC04] Shawe-Taylor, J. und Cristianini, N. *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press, 2004.

[Sti10] Stiefel, P. „Eine dezentrale Informations- und Kollaborationsarchitektur für die unternehmensübergreifende Produktentwicklung“. Diss. Technische Universität Clausthal, 2010.

[SWP00] Song, D. X., Wagner, D. und Perrig, A. „Practical techniques for searches on encrypted data“. In: *Security and Privacy. Proceedings. 2000 IEEE Symposium*. 2000, S. 44 –55.

[Tan02] Tanenbaum, A. *Computer Networks*. 4th. Prentice Hall Professional Technical Reference, 2002.

[ZMI91] Zheng, Y., Matsumoto, T. und Imai, H. „Structural Properties of One-Way Hash Functions“. In: *Advances in Cryptology — CRYPTO' 90 Proceedings*. Hrsg. von Menezes, A. und Vanstone, S. Bd. 537. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1991, S. 285–302.