UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Verteiltes Clustering für Multi-Source Entity Resolution gemischter Datensammlungen aus duplikatfreien und duplikatbehafteten Quellen

Masterarbeit

Leipzig, November 2020

vorgelegt von:

Stefan Lerm
Studiengang Informatik

Hochschullehrer: **Prof. Dr. Erhard Rahm**
Betreuer: **Alieh Saeedi, M.Sc.**

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

AP	Affinity Propagation
API	Programmierschnittstelle (application programming interface)
CLIP	Clustering based on LInk Priority
E/A	Ein-/Ausgabe
EAN	Europäische Artikelnummer
ER	Entity Resolution
FP	False Positive
HAP	Hierarchical Affinity Propagation
ID	Identifikationsschlüssel
MPN	Manufacturer Part Number
MS	Multi-Source
MSCD	Multi-Source Clean-Dirty
MSCD-AP	Multi-Source Clean-Dirty Affinity Propagation
MSCD-HAP	Multi-Source Clean-Dirty Hierarchical Affinity Propagation
POJO	Plain Old Java Object
SW	Schwellwert
TG	Teilgraph
TP	True Positive

Kapitel 1

Einleitung

Entity Resolution (ER) gewinnt zunehmend Bedeutung in der täglichen Arbeit von Unternehmen sowie von Forschungs- und Regierungsorganisationen [7, S. 1]. Wenn große Datenmengen aus verschiedenen Quellen zusammengeführt werden, wie beispielsweise in Vergleichsportalen, bei der Marktforschung oder bei der Fusion von Unternehmen, kommt häufig ein ER-System zum Einsatz. ER wird auch als *Deduplication*, *Record Linkage* oder *Object Matching* bezeichnet und beschreibt den Vorgang der Identifikation und Zuordnung von Beschreibungen, bei denen es sich um das gleiche Objekt der realen Welt handelt. Dies könnte beispielsweise das gleiche Produkt in verschiedenen Onlineshops sein, welches in beiden Datenbanken mit leicht unterschiedlichen Artikelinformationen eingepflegt ist. Im Big-Data-Umfeld kann die Anzahl der zusammenzuführenden Datenquellen beliebig skalieren. Man spricht dabei von Mulit-Source-ER. Deduplication kann aber auch innerhalb einer einzelnen Datenquelle durchgeführt werden, um diese von Duplikaten zu befreien. [38, S. 278 f.]

An der Universität Leipzig wurde mit *FAMER* (FAst Multi-source Entity Resolution system) [38] ein ER-System für das Big-Data-Umfeld entwickelt. Es ist in der Lage, große Datenmengen aus einer beliebigen Anzahl von Quellen parallel zu verarbeiten. Eine detaillierte Beschreibung von FAMER erfolgt in Abschnitt 3.3. Das System arbeitet in mehreren aufeinanderfolgenden Verarbeitungsschritten. Einer dieser Schritte ist das *Clustering*, also das Erzeugen von Gruppen zusammengehöriger Datensätze. Derzeit sind in FAMER sieben verschiedene Clustering-Algorithmen implementiert.

Bei Multi-Source-ER können die Datenquellen in verschiedener Qualität bezüglich des Vorhandenseins von Duplikaten vorliegen. Sie können selbst Duplikate von Datensätzen enthalten oder duplikatfrei sein. Die bisher in FAMER implementierten Clustering-Algorithmen können nur eingeschränkt verwendet werden. Einige sind für das Auffinden von Dubletten innerhalb einer einzelnen oder in mehreren duplikat-

behafteten Quellen ausgelegt. Andere treffen die Annahme, dass sämtliche Quellen duplikatfrei vorliegen. Dies kann erreicht werden, indem zuvor alle Datenquellen dedupliziert werden. Bei der potentiell großen Anzahl von Datenquellen im Big-Data-Umfeld kann dies einen hohen Aufwand bedeuten, der möglicherweise nicht immer erfolgreich ist [31, S. 9]. Mit einem Clustering-Algorithmus, der die Quellqualität bezüglich der Duplikatfreiheit berücksichtigt, kann dieser aufwändige Vorverarbeitungsschritt eingespart werden.

Ziel der vorliegenden Masterarbeit ist der Entwurf und die prototypische Implementierung eines Clustering-Verfahrens für Multi-Source-ER, welches für die gleichzeitige Verarbeitung von duplikatfreien und duplikatbehafteten Datenquellen eingesetzt werden kann. Der neu entworfene Clustering-Algorithmus trägt den Namen MSCD-AP und ist eine Erweiterung des etablierten Verfahrens Affinity Propagation (AP). Der Prototyp soll mit Hilfe des Frameworks *Apache Flink* entwickelt und in FAMER integriert werden.

Das nachfolgende Kapitel analysiert das Problem näher und unterteilt dazu ER in verschiedene Kategorien. In Kapitel 3 werden verwandte Arbeiten und Grundlagen vorgestellt, die für das Verständnis von ER sowie dem Basisalgorithmus AP und FAMER notwendig sind. Anschließend erarbeitet Kapitel 4 den konzeptuellen Entwurf von MSCD-AP sowie verschiedene Ansätze zur skalierbaren Gestaltung des Algorithmus. In Kapitel 5 werden die prototypischen Implementierungen der verschiedenen Ansätze von MSCD-AP ausführlich erläutert. Kapitel 6 widmet sich der vergleichenden Evaluation der Prototypen mit den übrigen, in FAMER enthaltenen Clustering-Verfahren. Abschließend werden die erzielten Ergebnisse in Kapitel 7 zusammengefasst und ein Ausblick über mögliche zukünftige Arbeit und Forschung gegeben.

Kapitel 2

Problemanalyse

Nachdem in Kapitel 1 eine Einführung in die Problemstellung gegeben wurde, erfolgen nun eine detaillierte Problemanalyse anhand des aktuellen Stands der Forschung sowie eine Definition der Anforderungen an eine Problemlösung. Dazu untergliedert Abschnitt 2.1 zunächst ER in verschiedene Kategorien, abhängig von der Duplikatfreiheit der aufzulösenden Quellen. Diesen Kategorien lassen sich spezialisierte Clustering-Algorithmen zuordnen. Für eine dieser Kategorien existiert derzeit noch kein passendes Clustering-Vefahren. Abschnitt 2.2 beschreibt die Anforderungen an einen Clustering-Algorithmus, für die bislang noch unberücksichtigte Kategorie. Abschließend grenzt Abschnitt 2.3 das Ziel der vorliegenden Masterarbeit näher ein.

2.1 Multi-Source Entity Resolution und Clustering

ER ist ein komplexer Vorgang zur Identifikation und Zuordnung von Beschreibungen, bei denen es sich um das gleiche Objekt der realen Welt handelt. In Abschnitt 3.1 erfolgt eine detaillierte Beschreibung von ER. Christophides et al. geben in [7] einen ganzheitlichen Überblick über den aktuellen Stand der Forschung für ER im Big-Data-Umfeld. Sie unterscheiden drei Typen von ER, abhängig von den Charakteristiken der zu verarbeitenden Datensammlungen:

1. **Clean-Clean-ER:** Die Eingabe besteht aus zwei duplikatfreien („sauberen“) Datenquellen. Das Ziel ist es, passende Paare zwischen beiden Quellen zu finden. Ein Objekt der ersten Quelle kann maximal einem Objekt der zweiten Quelle zugeordnet werden.

2. **Dirty-ER:** Die Eingabe besteht aus einer einzelnen „schmutzigen“ Datenquelle, die Duplikate von Datensätzen enthält. Ein Element kann beliebig oft dupliziert vorliegen. Ziel ist die Identifikation dieser Duplikate und die Bildung von Gruppen identischer Objekte.

3. **Multi-Source (MS) ER:** In der Eingabe sind mehr als zwei Quellen vorhanden. Es sollen Gruppen übereinstimmender Objekte aus allen Datenquellen gebildet werden.

Im Allgemeinen ist ER ein Vorgang aus einer Reihe von Teilschritten, bei denen das Clustering den Abschluss bildet. Das Clustering bildet disjunkte Gruppen der verarbeiteten Datensätze, so dass sämtliche Beschreibungen der gleichen Entität zu einer gemeinsamen Gruppe zugeordnet sind. Für jede repräsentierte Entität existiert dabei exakt ein Cluster. In Abschnitt 3.1.4 wird Clustering näher erläutert. Für jeden der drei ER-Typen existieren diverse Clustering-Algorithmen [7, S. 28 f.]. So werden für Clean-Clean-ER die Verfahren *Unique Mapping Clustering* und der *Kuhn-Munkres-Algorithmus* (auch *Ungarische Methode* genannt) vorgestellt. Für Dirty-ER führen Christophides et al. *Center Clustering*, *Merge-Center Clustering*, *Star Clustering*, *Ricochet Clustering*, *Markov Clustering*, *Cut Clustering* und *Correlation Clustering* auf. Die meisten der Algorithmen für Dirty-ER seien auch für Multi-Source-ER anwendbar. Es existieren aber auch dafür spezialisierte Verfahren, und zwar *SplitMerge* und *CLIP*.

Für Multi-Source-ER unterscheiden Christophides et al. jedoch nicht, ob die Quellen vollständig oder teilweise duplikatfrei vorliegen oder ob sämtliche Quellen duplikatbehaftet sind. Zur Berücksichtigung dieser Fälle muss das Schema folgendermaßen erweitert werden:

3. **Multi-Source-ER:**
    a) **MS-Clean-ER:** Sämtliche Datenquellen sind duplikatfrei. Eine Gruppe von Objekten der gleichen Entität (Cluster) darf maximal ein Objekt für jede Datenquelle enthalten.
    b) **MS-Dirty-ER:** Sämtliche Datenquellen enthalten Duplikate oder sind ungeprüft und können potentiell doppelte Datensätze enthalten. Ein Cluster kann mehrere Elemente aus jeder Quelle enthalten.
    c) **MS-Clean-Dirty (MSCD) ER:** Einige der Quellen sind duplikatfrei, andere können potentiell doppelte Datensätze enthalten. In jedem Cluster können duplikatbehaftete Quellen von mehreren Elementen, duplikatfreie Quellen aber nur von einem Objekt vertreten sein.

Bei erneuter Zuordnung der Clustering-Verfahren unter Berücksichtigung des neuen Schemas fällt auf, dass nicht sämtliche Fälle abgedeckt sind. SplitMerge und CLIP gehen von der Annahme aus, dass sämtliche Quellen duplikatfrei vorliegen. Sie können also nur für MS-Clean-ER eingesetzt werden. Würden die Algorithmen

für Quellen eingesetzt werden, die Duplikate enthalten, dann würden für eine Entität mehrere nicht überlappende Cluster entstehen. Das Anwenden der Algorithmen für Dirty-ER auf mehrere Quellen kann nur für MS-Dirty-ER sinnvoll erfolgen. Die Algorithmen arbeiten dann so, als ob sämtliche Datensätze aus einer Quelle stammen würden. Eine Unterscheidung der Quellen während des Clusterings wäre dann nicht erforderlich. Übrig bleibt jedoch der Fall MSCD-ER. Hierfür existieren noch keine speziellen Algorithmen. Diese Lücke soll mit der vorliegenden Masterarbeit gefüllt werden.

Mit bisherigen Mitteln ließe sich MSCD-ER nur in mehreren Durchläufen des ER-Prozesses umsetzen. Zunächst müssten alle duplikatbehafteten Datenquellen dedupliziert werden. Dies könnte entweder einzeln, durch jeweils einen separaten Dirty-ER Vorgang oder gemeinsam, mittels MS-Dirty-ER, erfolgen. Anschließend könnten die nun deduplizierten Datenquellen gemeinsam mit den von vornherein schon duplikatfreien Quellen durch MS-Clean-ER aufgelöst werden. Obraczka et al. experimentieren in [31] mit einem solchen Ansatz für eine Datensammlung, die ausschließlich aus duplikatbehafteten Quellen besteht. Dieser aufwändige Ansatz erzielt für die getestete Datensammlung jedoch schlechtere Ergebnisse als MS-Dirty-ER. Fehler bei der Deduplizierung einer Quelle im ersten Schritt pflanzen sich im MS-Clean-ER Vorgang des zweiten Schritts fort. Mit Hilfe des in der vorliegenden Arbeit anzufertigenden Clustering-Verfahrens soll MSCD-ER in einem einzelnen Durchlauf des ER-Prozesses ermöglicht werden.

2.2 Anforderungen an ein MSCD-Clustering-Verfahren

Ein Clustering-Algorithmus für die bisher noch unberücksichtigte ER-Kategorie MSCD-ER muss gewisse Anforderungen erfüllen, welche in diesem Abschnitt erläutert werden. Clustering ist typischerweise der letzte Schritt in einem ER-Prozess (vgl. Abschnitt 3.1). In den vorausgehenden Schritten werden einzelne Entitätsbeschreibungen miteinander verglichen, um ihre Ähnlichkeiten zu bestimmen. Diese Ähnlichkeiten können in Form eines Similarity-Graphen dargestellt werden. Sie bilden die Eingabe für das Clustering. Abbildung 2.2.1 zeigt einen beispielhaften Similarity-Graph für fünf Entitätsbeschreibungen. Er enthält einen Knoten für jede Beschreibung der aufzulösenden Datensammlung. Die Knoten sind in dem Beispiel kreisförmig dargestellt und weisen die Datenquelle (*src*) der Entitätsbeschreibung aus. Einige der Knoten sind durch gewichtete Kanten miteinander verbunden, wobei das Kantengewicht die paarweise Ähnlichkeit zwischen den Beschreibungen re-

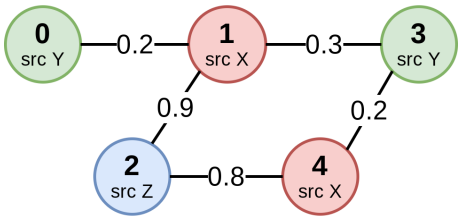
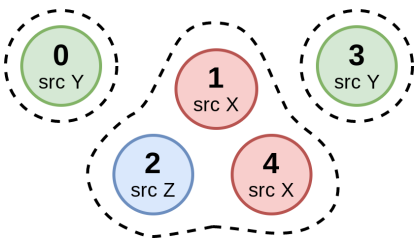


Abbildung 2.2.1: Beispielhafter Similarity-Graph

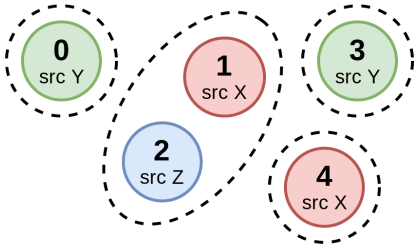
präsentiert. Die Kantengewichte befinden sich in einem Intervall von $[0, 1]$. 0 steht für keine und 1 für eine Übereinstimmung von 100%. Aufgrund von Blocking und Entity-Matching (vgl. Abschnitt 3.1.1 und Abschnitt 3.1.3) ist nicht zwischen jedem Knotenpaar eine Kante vorhanden.

Ein Clustering-Verfahren im Allgemeinen hat das Ziel, die Knoten so in Cluster zu gruppieren, dass die Ähnlichkeiten zwischen den Objekten innerhalb eines Clusters maximal und zwischen Objekten verschiedener Cluster minimal sind [38, S. 281]. Dazu wird versucht, neue Kanten aus indirekten Beziehungen zwischen den Knoten abzuleiten und bestehende Kanten zu verwerfen, wenn sie nicht zwei Objekte der gleichen Entität verbinden [7, S. 28].

Ein Clustering-Verfahren, welches für MSCD-ER eingesetzt werden kann, muss eine Restriktion berücksichtigen. Ein Cluster darf nicht mehrere Objekte der gleichen „sauberen“ Quelle enthalten. Dabei sollen die Informationen über die duplikatfreien Datenquellen so genutzt werden, dass eine möglichst optimale Gruppierung bezüglich des allgemeinen Clustering-Ziels entsteht. Wie auch bei den anderen ER-Varianten sind die Cluster im Ergebnis nicht überlappend. Ein Knoten darf also nicht mehreren Clustern zugeordnet sein.



(a) MSCD-Clustering für den Fall, dass Quelle X duplikatbehaftet ist



(b) MSCD-Clustering für den Fall, dass Quelle X duplikatfrei ist

Abbildung 2.2.2: *MSCD-Clustering für den beispielhaften Similarity-Graph in Abbildung 2.2.1.* Alle Punkte, die von einer gemeinsamen gestrichelten Linie umgeben sind, befinden sich im gleichen Cluster.

Der beispielhafte Similarity-Graph auf Abbildung 2.2.1 enthält die beiden Knoten eins und vier, welche beide aus der gleichen Quelle X stammen und eine sehr hohe Ähnlichkeit zu Knoten zwei aus Quelle Z aufweisen. Falls Quelle X duplikatbehaftet ist, sollte ein MSCD-Clustering-Verfahren eine transitive Relation zwischen den Knoten eins und vier herleiten. Das bedeutet, aus der gemeinsamen hohen Ähnlichkeit zu Knoten zwei kann der Algorithmus folgern, dass Beschreibung eins und vier ebenfalls eine hohe Übereinstimmung zueinander aufweisen. Das Clustering würde dann wie in Abbildung 2.2.2a dargestellt erfolgen. Ist Quelle X jedoch duplikatfrei, müsste ein MSCD-Clustering-Algorithmus eine Trennung der Datensätze eins und vier sicherstellen. Er würde also die Beschreibungen entsprechend Abbildung 2.2.2b gruppieren. Die Eigenschaft eines Clusterings, Datensätze der gleichen duplikatfreien Quelle vollständig zu separieren, erhält die Bezeichnung Clean-Source-Konsistenz. Im weiteren Verlauf der Arbeit wird immer wieder auf dieses Einführungsbeispiel zurückgegriffen, um diverse Konzepte zu erläutern.

2.3 Abgrenzung

Ziel dieser Masterarbeit ist der Entwurf eines für MSCD-ER spezialisierten Clustering-Algorithmus. Das Verfahren soll prototypisch implementiert und in das bestehende ER-System *FAMER* integriert werden. Um die Skalierbarkeit für große Datenmengen und eine Kompatibilität zu FAMER sicherzustellen, setzt die Implementierung, ebenso wie FAMER, auf dem Framework *Apache Flink* auf. Die übrigen Schritte und Aspekte von ER sowie die Komponenten von FAMER werden zum besseren Verständnis des Gesamtprozesses erläutert. Es findet jedoch keine Optimierung anderer Komponenten hinsichtlich MSCD-ER statt. Die Arbeit beschränkt sich ausschließlich auf das Clustering, wobei von einem Similarity-Graphen als Eingabe ausgegangen wird. Zur Bestimmung des Clusterings kann der zu entwerfende Algorithmus nur über die folgenden Informationen verfügen:

- die vorab bestimmten, paarweisen Ähnlichkeiten zwischen den Entitätsbeschreibungen,
- die Struktur des Similarity-Graphen,
- eine Kennzeichnung für jede Beschreibung, aus welcher Datenquelle sie stammt
- und eine Angabe darüber, welche der Quellen duplikatbehaftet und welche duplikatfrei vorliegen.

Kapitel 3

Grundlagen und verwandte Arbeiten

Nachdem im vorangegangenen Kapitel die Problemstellung des Entwurfs und der Implementierung eines Clustering-Verfahrens für MSCD-ER vorgestellt, analyisiert und abgegrenzt wurde, schafft das folgende Kapitel die für dessen Umsetzung nötigen Grundlagen. Dazu erfolgt in Abschnitt 3.1 zunächst eine ausführliche Erläuterung von ER anhand der üblichen Schritte eines vollständigen ER-Prozesses. Dabei wird besonders auf jene Komponenten und Methoden eingegangen, die später bei der Evaluation in Kapitel 6 Verwendung finden. Hervorzuheben sind die Technologien zur Erstellung eines Similarity-Graphen sowie alle in *FAMER* enthaltenen Clustering-Algorithmen. Anschließend stellt Abschnitt 3.2 das verteilte System *Apache Flink* vor, auf welchem FAMER aufbaut. Der Abschnitt geht besonders auf die Architektur und die für die Implementierung der Prototypen relevanten Programmierschnittstellen ein. Der aktuelle Stand von FAMER wird in Abschnitt 3.3 präsentiert und erläutert. Abschnitt 3.4 widmet sich dem Basisalgorithmus Affinity Propagation (AP), auf dem das neu entworfene Clustering-Verfahren für MSCD-ER aufbaut. Die letzten Abschnitte 3.5 und 3.6 stellen schließlich zwei Abwandlungen von AP vor, Adaptive AP und Hierarchical AP, die eine fehlertolerante und skalierbare Implementierung des Verfahrens ermöglichen.

3.1 Entity Resolution

Schubert definiert eine Entität, im Kontext der Modellierung von Datenstrukturen, als einen (systemrelevanten) Ausschnitt der Realität, der sich aufgrund seiner Eigenschaften eindeutig identifizieren lässt [41, S. 276]. Eine solche Entität ist also ein bestimmtes Objekt der realen Welt, wie beispielsweise eine Person, ein bestimmtes Produkt, wie eine Digitalkamera oder auch ein abstraktes Konstrukt, wie ein Konto. Die gleiche Entität kann in verschiedenen Datenquellen beschrieben sein. So bieten beispielsweise verschiedene Online-Shops das gleiche Modell einer Digitalkamera an

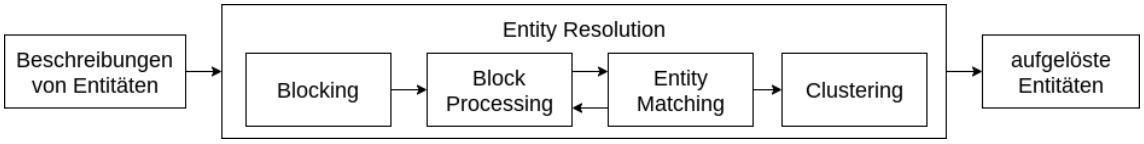


Abbildung 3.1.1: Der grundlegende ER-Prozess [7, S. 5]

oder halten mehrere Unternehmen Daten zur selben Person in ihren Kundendatenbanken vor.

Viele Organisationen, wie Unternehmen oder staatliche Einrichtungen, müssen regelmäßig große Mengen von Daten aus verschiedenen internen und externen Quellen zusammenführen. Dieser Prozess wird als Datenintegration bezeichnet. ER ist dabei eine wichtige Teilaufgabe, die es ermöglicht, verschiedene Beschreibungen der gleichen Entität zu identifizieren, wenn kein eindeutiger Identifikationsschlüssel vorhanden ist. Dabei müssen diverse Qualitätsprobleme der Datenquellen bewältigt werden, wie unvollständige, redundante, inkonsistente oder auch falsche Daten. [7, S. 1]

Christophides et al. beschreiben in [7] die grundlegenden Verarbeitungsschritte, die im Allgemeinen für einen vollständigen ER-Prozess notwendig sind (vgl. Abb. 3.1.1). Die Eingabe für einen ER-Prozess sind Entitätsbeschreibungen. Diese werden über mehrere Schritte verarbeitet, um abschließend gruppiert als aufgelöste Entitäten ausgegeben zu werden. Jede der disjunkten Gruppen enthält also Beschreibungen der gleichen Entität, wobei für jede Entität jeweils genau eine Gruppe vorhanden sein sollte. Der Prozess besteht aus den aufeinanderfolgenden Schritten Blocking, Block-Processing, Entity-Matching und Clustering, die im Folgenden näher erläutert werden. Zur Veranschaulichung dienen dabei die beispielhaften Produktbeschreibungen von Digitalkameras in Tabelle 3.1.1. Für dieses einfache Beispiel wird von Dirty-ER ausgegangen. Alle Beschreibungen stammen aus der gleichen, nicht duplikatfreien Datenquelle.

Name	Herstellername	Modellbezeichnung	EAN	Preis [€]
a	Nikon	P7100		329,99
b	Nikon	P-7100	0018208924127	337,99
c	Nikon		0018208924127	
d	Nikon	S9300	0018208263158	113,99
e	Canon	A3300 IS	0013803133936	109,95
f	Canon	PowerShot A3300 IS		159.99

Tabelle 3.1.1: *Beispiel für Produktbeschreibungen von Digitalkameras.* EAN steht für die europäische Artikelnummer und ist eine international eindeutige Produktkennzeichnung.

3.1.1 Blocking

Damit die Beschreibungen der gleichen Entität identifiziert werden können, müssen diese miteinander verglichen werden. Ein übereinstimmendes Beschreibungspaar wird als *Match* bezeichnet. Wenn beispielsweise beim Dirty-ER (vgl. Abschnitt 2.1) jede Entitätsbeschreibung detailliert mit jeder anderen verglichen werden müsste, wären bei N Beschreibungen $\frac{N^2-N}{2}$ Vergleiche notwendig. Das sogenannte *Blocking* dient der Reduktion der notwendigen Vergleiche und ist üblicherweise der erste Schritt eines ER-Prozesses. Dafür werden ähnliche Beschreibungen anhand eines bestimmten Kriteriums in Blöcke gruppiert. Dieses Kriterium wird als *Blocking-Key* bezeichnet. Er lässt sich auf vielfältige Weise aus den Attributwerten der Entitätsbeschreibungen erzeugen. So können beispielsweise Präfixe der Werte verschiedener Attribute kombiniert oder ein phonetischer Index wie der *Soundex* berechnet werden. Letzterer bildet ähnlich klingende Worte auf einen gemeinsamen Schlüssel ab. [5, S. 69 f.]

Nach der Gruppierung durch das Blocking müssen nur noch die Beschreibungen miteinander verglichen werden, die sich im gleichen Block befinden. Dabei besteht jedoch die Gefahr Matches zu „verlieren“, wenn übereinstimmende Beschreibungen nicht dem gleichen Block zugeteilt werden. Die Ziele des Blockings sind es also:

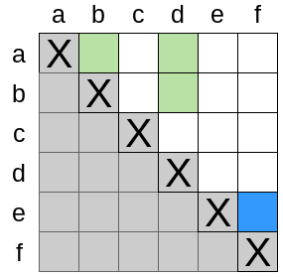
1. die Zahl der nötigen Vergleiche so weit wie möglich zu reduzieren
2. und alle Paare von übereinstimmenden Beschreibungen dem gleichen Block zuzuordnen. [7, S. 6]

Diese beiden Ziele konkurrieren, da das erste Ziel möglichst kleine und das zweite Ziel möglichst große Blöcke erfordert. Verschiedene Blocking-Strategien können eines der beiden Ziele priorisieren oder einen Mittelweg wählen. Christophides et al. führen 21 verschiedene Blocking-Methoden auf [7, S. 13]. Zum besseren Verständnis der Möglichkeiten werden im Folgenden die beiden Verfahren *Standard-Blocking* und *Sorted-Neighborhood* für das Beispiel aus Tabelle 3.1.1 vorgestellt.

Beim Standard-Blocking wird jede Beschreibung anhand ihres Blocking-Keys exakt einem Block zugeordnet. Alle Beschreibungen mit dem gleichen Schlüssel befinden sich im gleichen Block. Im Beispiel auf Abbildung 3.1.2 bilden die ersten drei Zeichen des Herstellernamens den Blocking-Key. Dadurch sind die Beschreibungen a, b und d dem ersten Block und e und f dem dritten Block zugeordnet. Vergleiche müssen nur für die in Abb. 3.1.2b farbig markierten Beschreibungspaare durchgeführt werden. Die Berechnungen für die weiß markierten Paare erspart das Standard-Blocking. Graue Felder stehen für redundante und Selbst-Vergleiche. Beschreibung c

Blocking-Key	Beschreibungen
nik	a, b, d
nic	c
can	e, f

(a) Blockzuordnungen entsprechend des Blocking-Keys aus den ersten drei Zeichen des Herstellernamens



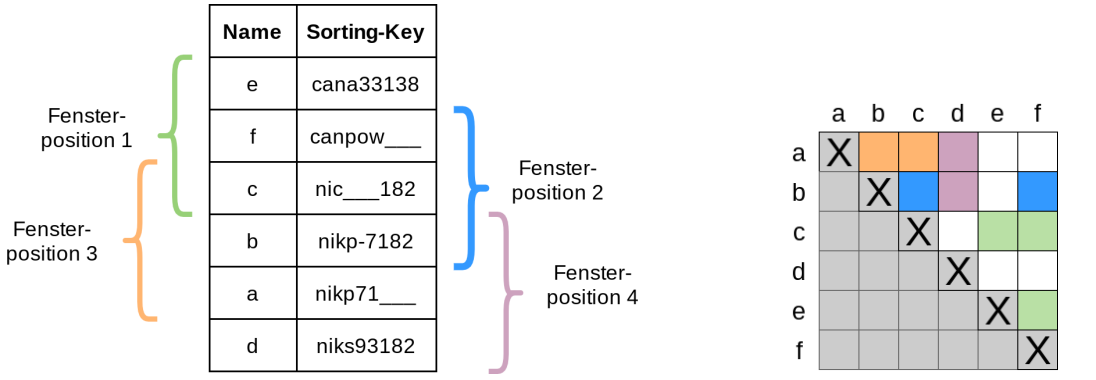
(b) Auszuführende Vergleiche

Abbildung 3.1.2: Standard-Blocking bei Dirty-ER

ist durch den fehlerhaften Herstellernamen einem separaten Block zugewiesen und kann im restlichen ER-Prozess nicht mehr mit den eigentlich übereinstimmenden Beschreibungen a und b verglichen werden. Eine effiziente Implementierung des Verfahrens ist mit der Datenstruktur des invertierten Index möglich. Dabei dient der Blocking-Key als Index-Schlüssel. Für jeden Index-Schlüssel wird eine Liste der jeweiligen Identifikationsschlüssel (IDs) der Entitätsbeschreibungen gespeichert, die diesem zugeordnet sind. [5, S. 80 f.]

Beim Sorted-Neighborhood-Blocking von Hernandez und Stolfo [20, 21] kann eine Beschreibung mehreren Blöcken zugeordnet werden. Dazu verwendet es ein sogenanntes *Sliding Window* der Größe $w > 1$. Anstelle von Blocking-Keys erfolgt das Blocking mit Hilfe von sogenannten Sorting-Keys. Letztere lassen sich ebenso aus den Attributwerten der Entitätsbeschreibungen erzeugen. Allerdings gelten bei der Definition des Sorting-Keys andere Kriterien, denn es werden nicht die Elemente mit identischem Schlüssel gruppiert, sondern jene, deren Schlüssel-Werte bei Sortierung in das gleiche Sliding-Window fallen. Dazu werden die Sorting-Keys aller Beschreibungen erzeugt und anschließend sortiert.

Im Beispiel auf Abbildung 3.1.3 bildet sich der Sortierschlüssel aus den jeweils drei Zeichen umfassenden Präfixen der Attribute Herstellername, Modellbezeichnung und EAN. Die Fenstergröße beträgt $w = 3$. Das Fenster wird über die sortierten Schlüssel geschoben. Sämtliche Paare von Entitätsbeschreibungen, deren Schlüssel in eine gemeinsame Position des Sliding-Windows fallen, bilden sogenannte Kandidaten-Paare. So formen die Elemente e, f und c einen Block für die erste Position des Sliding-Windows. f, c und b fallen in einen Block für die zweite Fensterposition. In Abb. 3.1.3b sind die bei jeder Verschiebung des Fensters neu hinzugekommenen Kandidaten-Paare mit der Farbe der jeweiligen Fensterposition gekennzeichnet. Anders als beim Standard-Blocking kann die fehlerhafte Beschreibung c beim Sorted-Neighborhood-Blocking mit den passenden Elementen a und



(a) Fensterpositionen des Sliding-Windows der Größe $w = 3$

(b) Auszuführende Vergleiche

Abbildung 3.1.3: Sorted-Neighborhood-Blocking bei Dirty-ER

b verglichen werden. Viele Beschreibungspaare tauchen in mehreren Blöcken auf, was zu unnötigen redundanten Vergleichen führen würde. Der nächste Schritt des ER-Prozesses, das sogenannte *Block-Processing*, dient dazu, diese wiederholten Vergleiche zu vermeiden. Um noch mehr Paare übereinstimmender Beschreibungen gemeinsamen Blöcken zuzuordnen, schlagen Hernandez und Stolfo in [21] das Multi-Pass-Blocking vor. Dabei wird das Blocking für verschiedene Schlüssel wiederholt ausgeführt. [5, S. 81 ff.]

3.1.2 Block-Processing

Viele Blocking-Verfahren, wie beispielsweise Sorted-Neighborhood-Blocking, verwenden überlappende Blöcke, bei denen eine Entitätsbeschreibung in mehreren Blöcken vertreten sein kann. Dadurch können auch Paare zweier Beschreibungen mehrfach vorkommen und müssten redundant verglichen werden. Das Block-Processing vermeidet sowohl diese wiederholten Vergleiche als auch unnötige Vergleiche von Paaren, die kein Match bilden. Christophides et al. unterscheiden drei verschiedene Techniken des Block-Processing [7, S. 14]:

1. **Blockzentrierte Methoden** basieren auf groben Charakteristiken der Blöcke. So werden beispielsweise beim *Block-Purging* einfach Blöcke verworfen, deren Größe einen Schwellwert überschreitet. Das *Block-Pruning* verarbeitet die Blöcke sortiert nach ihrer Größe. Es beginnt mit dem Kleinsten und beendet die Verarbeitung, wenn die Kosten zum Auffinden weiterer Matches einen Grenzwert übersteigen.

2. **Entitätszentrierte Methoden**. Für diese Kategorie existiert bisher nur das Verfahren des *Block-Filtering*. Es geht von der Annahme aus, dass größere Blöcke eine Entitätsbeschreibung schlechter repräsentieren als kleinere. Daher entfernt es jede Beschreibung aus dem größten Block, in dem sie vertreten ist.

3. **Vergleichszentrierte Methoden** entscheiden für die einzelnen Beschreibungspaare, ob diese verglichen werden müssen oder nicht. Ein Beispiel ist die *Comparison Propagation*, die sich die bereits verglichenen Paare merkt und nicht wiederholt vergleicht, wenn sie in einem anderen Block erneut auftauchen. *Meta-Blocking* Verfahren ermöglichen es außerdem, unnötige Vergleiche zu vermeiden. Anhand von Block-Charakteristiken wird ein Gewicht bestimmt, welches angibt, wie wahrscheinlich das Beschreibungspaar ein Match ergibt. Dazu können beispielsweise die Anzahl der gemeinsamen Blöcke und die Größe dieser gehören. Abhängig von dem berechneten Gewicht lässt sich wiederum mit verschiedenen Methoden eine Entscheidung treffen, ob ein detaillierter Vergleich für das Paar notwendig ist.

3.1.3 Entity-Matching

Der dritte Schritt eines ER-Prozesses ist das *Entity-Matching*. Jedes Beschreibungspaar, für das laut Block-Processing ein detaillierter Vergleich notwendig ist, wird mittels einer Ähnlichkeitsfunktion miteinander verglichen. Auf Basis des Ergebnisses bestimmt die Match-Funktion, ob die Beschreibungen zur gleichen Entität gehören. Die Ausgabe des Entity-Matchings ist der sogenannte *Similarity-Graph*. Dieser enthält einen Knoten für jede Entitätsbeschreibung und eine Kante zwischen jedem Knotenpaar, dessen Beschreibungen ein Match bilden. Die Ähnlichkeit zwischen beiden Knoten bildet das Kantengewicht. Abbildung 3.1.4 zeigt einen Similarity-Graph, der mit Hilfe der im Folgenden vorgestellten Methoden erzeugt wurde. [7, S. 18]

Die Ähnlichkeitsfunktion lässt sich im einfachsten Fall aus der Kombination verschiedener Attributähnlichkeiten definieren. So kann beispielsweise das Maß der Übereinstimmung zweier Beschreibungen von Digitalkameras aus der Kombination der Ähnlichkeiten für die Attribute „Herstellername“, „Modellbezeichnung“ und „Preis“ bestimmt werden. Die Attribute können entsprechend ihrer Aussagekraft unterschiedlich gewichtet sein. Dabei sei w_i das Gewicht von Attribut i. Der Grad der Übereinstimmung der Werte von Beschreibung a und b für das jeweilige Attribut wird mit $sim_i(a, b)$ bezeichnet. Die gewichtete Ähnlichkeit $wsim(a, b)$ über alle Attribute (att), die bei beiden Beschreibungen vorhanden sind, ergibt sich entsprechend Formel 3.1.1.

Attribut	Gewicht	Beschr. a	Beschr. b	Ähnl.	Grenzw.	Match
Herstellername	2	Nikon	Nikon	1,0		
Modellbezeichnung	3	P7100	P-7100	0,83		
EAN	4		0018208924127	-		
Preis [€]	1	329,99	337,99	0,98		
				0,91	0,85	ja

Tabelle 3.1.2: Beispiel für die gewichtete Ähnlichkeit zweier Beschreibungen von Digitalkameras

$$wsim(a,b) = \frac{\sum_{i \in att} w_i \cdot sim_i(a,b)}{\sum_{i \in att} w_i} \tag{3.1.1}$$

Tabelle 3.1.2 zeigt dies anhand eines Beispiels. Die Match-Funktion ist im einfachsten Fall durch einen Schwellwert definiert. Übersteigt die gewichtete Ähnlichkeit den Grenzwert, dann handelt es sich bei dem Beschreibungspaar um ein Match.

Für die einzelnen Attribute kommen je nach Datentyp unterschiedliche Vergleichsverfahren zum Einsatz. Ein Maß für die Übereinstimmung zweier Zeichenketten $s1$ und $s2$ ist zum Beispiel die Levenshtein-Distanz ($dist_{lev}$). Dabei wird die minimale Anzahl von Veränderungen ermittelt, um die erste Zeichenfolge in die zweite umzuwandeln. Entsprechend Formel 3.1.2 lässt sich damit die Levenshtein-Ähnlichkeit bestimmen, wobei der Betrag einer Zeichenkette für ihre Länge steht. Für die beiden beispielhaften Modellbezeichnungen in Tabelle 3.1.2 beträgt $sim_{lev} = 0,83$. [5, S. 103 f.]

$$sim_{lev}(s1,s2) = 1 - \frac{dist_{lev}(s1,s2)}{max(|s1|,|s2|)} \tag{3.1.2}$$

Zwei weitere Maße für die Ähnlichkeit zweier Zeichenfolgen sind der *Dice-* und der *Jaccard-Koeffizient*. Für beide werden $s1$ und $s2$ zunächst in sogenannte Q-Gramme zerlegt. Q-Gramme sind Teilworte des Ursprungsworts der Länge Q. So ergeben sich beispielsweise aus der Modellbezeichnung „P7100“ die 3-Gramme „P71“, „710“ und „100“. Die Bezeichnung „P-7100“ von Beschreibung b zerlegt sich in „P-7“, „-71“, „710“ und „100“. Der Dice-Koeffizient ergibt sich entsprechend Formel 3.1.3 mit $c1$ als Anzahl der Q-gramme von $s1$, $c2$ jener von $s2$ und c_{common} als Anzahl der Q-Gramme, die in beiden Zeichenketten gleichermaßen vorkommen. Der Jaccard-Koeffizient verwendet die Formel 3.1.4. Für die beiden beispielhaften Modellbezeichnungen beträgt $sim_{dice3} = \frac{4}{7}$ und $sim_{jaccard3} = \frac{2}{5}$. Optional können spezielle zusätzliche Zeichen für Wortanfang und -ende in die Q-Gramme aufgenommen werden. Es existieren viele weitere Maße für die Ähnlichkeit von Zeichenfolgen, wie beispielsweise die *Jaro-Winkler-Ähnlichkeit*, die speziell für den Vergleich von Namen entwickelt wurde. [5, S. 106 ff.]

$$sim_{diceQ}(s1, s2, Q) = \frac{2 \cdot c_{common}}{c1 + c2} \tag{3.1.3}$$

$$sim_{jaccardQ}(s1, s2, Q) = \frac{c_{common}}{c1 + c2 - c_{common}} \tag{3.1.4}$$

Für numerische Attribute, wie die Verkaufspreise zweier Kameras, kann eine Ähnlichkeit über den Betrag der Differenz beider Werte ($n1$ und $n2$) bzw. des relativen Anteils dieses Preisabstands am größeren der beiden Preise bestimmt werden (vgl. Formel 3.1.5 [5, S. 121]). Auch für weitere Attributtypen, wie geographische Koordinaten oder Datumsangaben, existieren spezielle Ähnlichkeitsmaße.

$$sim_{num}(n1, n2) = 1 - \frac{|n1 - n2|}{max(|n1|, |n2|)} \tag{3.1.5}$$

Die Match-Funktion kann weit komplexer als ein einfacher Schwellwert sein. Mehrere Bedingungen für einzelne oder Paare von Attributähnlichkeiten können definiert und kombiniert werden. So könnte beispielsweise eine Regel vorschreiben, dass beide Beschreibungen entweder über eine Modellbezeichnung oder eine EAN verfügen müssen und dass die Summe aus der Ähnlichkeit von Herstellername und Modellbezeichnung einen bestimmten Wert überschreiten muss. Solche Regeln lassen sich manuell definieren oder mit Hilfe von Trainingsdaten über ein maschinelles Lernverfahren ermitteln.

Eine Kombination aus Ähnlichkeits- und Match-Funktion, die für alle Beschreibungspaare perfekt unterscheidet, ob es sich um ein Match handelt oder nicht, ist sehr schwer zu finden. In der Realität muss mit unvollständigen, inkonsistenten und falschen Daten gearbeitet werden. Als Beispiel sollen die Produktbeschreibungen a, b und c aus Tabelle 3.1.1 dienen. Obwohl alle Beschreibungen der gleichen Entität angehören, lassen sich a und c mit den gegebenen Informationen einander nicht zuordnen. Beschreibung a verfügt über keine EAN und c über keine Modellbezeichnung. Der Herstellername ist die einzige Übereinstimmung beider Elemente.

Sogenannte *Collective-ER*-Techniken gehen über den paarweisen Vergleich von Beschreibungen hinaus und beurteilen ihre Ähnlichkeit in einem iterativen Verfahren. So würde in einer ersten Iteration ein Match aus Beschreibung b und c gebildet. Die Attribute beider Beschreibungen können nun zusammengefügt werden, so dass c die Modellbezeichnung von b übernimmt. In der zweiten Iteration findet sich durch die neuen Informationen ein Match aus a und c. Ein solches iteratives Verfahren wird als *mergebasiert* bezeichnet. *Beziehungsbasierte* Verfahren verändern die Beschreibungen nicht, sondern reichen die Ähnlichkeitsinformationen aus der vorangegangenen Iteration zwischen Beschreibungen weiter, die ein Match gebildet haben. So kann

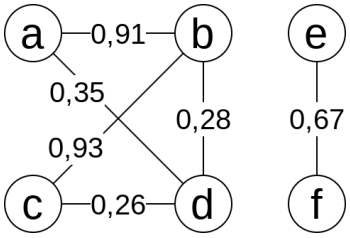


Abbildung 3.1.4: Similarity-Graph für die Beispieldaten aus Tabelle 3.1.1

c in der zweiten Iteration aus der Ähnlichkeit zwischen a und b folgern, dass es ebenfalls ein Match mit a bildet. [7, S. 19]

In einem iterativen ER-Prozess ist es möglich, Blocking und Matching miteinander zu verzahnen (vgl. Abb. 3.1.1). Neue Attributwerte aus einem mergebasierten Entity-Matching-Schritt können dafür verwendet werden, die Blockzuordnung der Beschreibungen zu verändern. Die veränderten Blöcke führen wiederum zu neuen Vergleichen im Matching-Schritt. Blocking und Matching wiederholen sich abwechselnd, bis sich die Blöcke nicht mehr verändern oder ein anderes Abbruchkriterium erreicht ist. [7, S. 5]

Abbildung 3.1.4 zeigt den für die Beispieldaten vom Entity-Matching ausgegebenen Similarity-Graph, welcher sich bei Verwendung von *Sorted-Neighborhood-Blocking* und einer Berechnung der gewichteten Ähnlichkeit entsprechend Tabelle 3.1.2 ergibt. Dabei wird für die Ähnlichkeit der EAN ein Wert von $1,0$ angenommen, wenn beide Nummern identisch sind und ein Wert von $0,0$, sobald sie sich an einer Stelle unterscheiden. Das Maß für die Übereinstimmung von Zeichenketten ist die *Levenshtein-Ähnlichkeit*. Der Grenzwert für die Match-Funktion beträgt $0,25$.

3.1.4 Clustering

Der letzte Schritt eines ER-Prozesses ist das Clustering. Die Eingabe besteht aus dem Similarity-Graphen, der zuvor beim Entity-Matching erzeugt wurde. Das Clustering gruppiert die Entitätsbeschreibungen, sodass möglichst alle Beschreibungen der gleichen Entität eine Gruppe bilden. Diese disjunkten Gruppen werden als Cluster bezeichnet und stellen die Ausgabe des Clustering-Schritts dar. Da die tatsächliche Entität einer Beschreibung nicht bekannt ist, versucht ein Clustering-Verfahren die Beschreibungen so zu gruppieren, dass die Ähnlichkeiten zwischen Elementen der gleichen Gruppe maximal und zwischen Elementen verschiedener Gruppen minimal sind. Dazu leitet es neue Kanten aus indirekten Beziehungen zwischen den Knoten des Similarity-Graphen her und verwirft bestehende Kanten, wenn diese mit großer Wahrscheinlichkeit nicht zwei Beschreibungen der gleichen Entität verbinden. [38, S. 281]

Das zu verwendende Clustering-Verfahren richtet sich nach der Problemstellung. In Abschnitt 2.1 wurde ER, abhängig von den Charakteristiken der zu verarbeitenden Datensammlung, in verschiedene Typen untergliedert, denen spezielle Clustering-Algorithmen zugeordnet wurden. Mit einigen dieser Verfahren erfolgt in Kapitel 6 eine vergleichende Evaluation mit dem neu entwickelten Clustering-Verfahren für MSCD-ER. Aufgrund dessen sollen sie an dieser Stelle kurz vorgestellt werden.

Connected Components: Hierbei handelt es sich um das einfachste Clustering-Verfahren. Es bildet jeweils ein Cluster für jede Gruppe aus Entitätsbeschreibungen, die direkt über Kanten oder indirekt durch transitive Relationen miteinander verbunden sind [37, S. 66 f.]. Kantengewichte werden dabei nicht berücksichtigt. So entstehen beispielsweise aus dem in Abbildung 3.1.4 dargestellten Similarity-Graphen zwei Cluster, eines mit den Beschreibungen a, b, c und d sowie eines bestehend aus e und f.

Center und Merge-Center: Im Gegensatz zu *Connected Components* verwendet das *Center*-Verfahren [19] die Kantengewichte des Similarity-Graphen. Es sortiert alle Kanten in absteigender Reihenfolge nach ihrem Gewicht und verarbeitet sie nacheinander. Sind beide Knoten v_i und v_j der Kante $E(v_i, v_j)$ noch keinem Cluster zugeordnet, dann bildet einer der beiden ein neues Cluster-Zentrum und der andere wird diesem zugewiesen. Falls v_i bereits Zentrum eines Clusters und v_j noch unzugewiesen ist, dann weist der Center-Algorithmus letzteren dem Cluster von v_i zu. In allen anderen Fällen ignoriert das Verfahren die Kante und setzt seine Arbeit mit der nächsten fort. Die Variante *Merge-Center* [19] iteriert ein zweites Mal über die sortierte Kantenliste. Dabei fügt es zwei Cluster zusammen, falls v_i ein Cluster-Zentrum und v_j bereits einem anderen Cluster zugeordnet ist. [37, S. 67 ff.]

Star Clustering: Für *Star Clustering* [1] existieren zwei verschiedene Varianten, die *Star-1* und *Star-2* genannt werden. Der Algorithmus berechnet initial den Grad eines jeden Knotens. In Star-1 ergibt sich dieser aus der Anzahl der Kanten eines Knotens. Star-2 berechnet den Knotengrad als Durchschnitt der Gewichte aller Kanten, mit denen ein Knoten verbunden ist. Im weiteren Verlauf arbeiten beide Varianten identisch. Der Algorithmus arbeitet iterativ und ermittelt in jeder Iteration den Knoten mit dem größten Grad, der noch keinem Cluster zugeordnet ist. Dieser Knoten bildet ein Cluster-Zentrum, dem alle Nachbarknoten zugeordnet werden. Bei gleichen Knotengraden entscheidet eine initial zufällig festgelegte Knotenpriorität. Das Verfahren endet, wenn alle Knoten einem Cluster zugeordnet sind. Im

Gegensatz zu den bisher vorgestellten Clustering-Algorithmen können hierbei überlappende Cluster entstehen. In einer zusätzlichen Nachverarbeitung muss für jeden Knoten, der mehreren Zentren zugeordnet ist, das beste gewählt werden. [37, S. 69]

CCPivot Correlation Clustering: Der parallele *CCPivot*-Algorithmus [4] ist eine Approximation für das Correlation Clustering [2] Verfahren. Es weist jedem Knoten zunächst eine zufällige Priorität zu. Anschließend verkleinert es den Similarity-Graphen in einem iterativen Verfahren Stück für Stück, indem es Knoten entfernt, denen es ein Cluster zugewiesen hat. Dazu ermittelt CCPivot in jeder Iteration den maximalen Knotengrad des verbliebenen Graphen. Aus den Knoten werden zufällig Kandidaten gewählt, die ein Cluster-Zentrum bilden sollen. Die Wahrscheinlichkeit für die Wahl zum Kandidaten ist dabei invers abhängig von dem maximalen Knotengrad sowie von einem Parameter ϵ. Je größer ϵ und je kleiner der maximale Knotengrad, desto wahrscheinlicher wird ein Knoten zum Kandidaten. Falls ein Kandidat über keine Kante zu einem anderen Kandidaten verfügt, dann bildet dieser ein Cluster-Zentrum. Dies verhindert überlappende Cluster. Alle benachbarten Knoten eines Cluster-Zentrums werden diesem zugewiesen. Verfügt ein Knoten über Kanten zu mehreren Zentren, dann wählt er jenes mit der höheren Priorität. Die Cluster-Zentren und ihre zugeordneten Knoten werden aus dem Similarity-Graph entfernt und die nächste Iteration beginnt. Der maximale Knotengrad des Graphen wird mit voranschreitender Iterationsanzahl kleiner, wodurch die Anzahl der gefundenen Cluster-Zentren zunimmt. Das Verfahren terminiert, wenn alle Knoten einem Cluster zugeordnet sind. [37, S. 69 f.]

CLIP: *Clustering based on LInk Priority* (CLIP) [40] ist ein spezielles Clustering-Verfahren für Multi-Source-Clean-ER. Es verwendet Kantencharakteristika und Informationen über die Datenquellen der Knoten, um die Clean-Source-Konsistenz des Clustering-Resultats zu garantieren. Eine Entitätsbeschreibung aus einer Quelle kann im Similarity-Graphen mit mehreren Beschreibungen einer anderen Quelle verbunden sein. Abbildung 3.1.5 zeigt den Similarity-Graph des Einführungsbeispiels, ergänzt um eine zusätzliche Kante zwischen Knoten 0_Y und 4_X. In dem Graphen ist beispielsweise die Entitätsbeschreibung 4_X aus Quelle X mit den Beschreibungen 0_Y und 3_Y aus Quelle Y sowie 2_Z aus Quelle Z verbunden. Die Kante, welche den Knoten 4_X mit dem ähnlichsten Punkt aus Quelle Y verbindet, wird in CLIP als maximaler Link bezeichnet. Im Beispiel ist dies die Kante $E(4_X, 0_Y)$ zu Knoten 0_Y. Gleichzeitig ist die Kante auch ein maximaler Link für Knoten 0_Y, doch dies muss nicht immer so sein. Wäre die Ähnlichkeit zwischen Beschreibung 0_Y und 1_X größer,

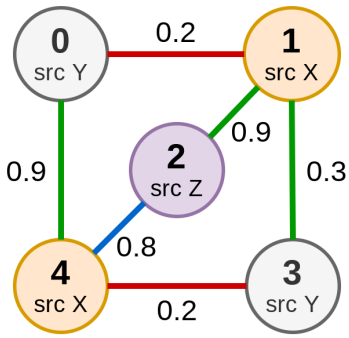


Abbildung 3.1.5: Similarity-Graph mit Kennzeichnung der Kantenstärken des CLIP-Algorithmus

dann wäre $E(0_Y, 1_X)$ anstatt dessen die maximale Verbindung von 0_Y zur Quelle X. Mit Hilfe der Definition des maximalen Links lassen sich drei verschiedene Kantenstärken bestimmen. Ein *Strong-Link* (grün dargestellt) ist eine Kante, welche für beide Knoten einen maximalen Link repräsentiert. *Normal-Links* (blau) sind nur für einen und *Weak-Links* (rot) für keinen der beiden Knoten maximal. Ein weiteres Kantencharakteristikum ist der Kantengrad. Er ergibt sich aus dem Minimum der Grade beider Knoten, welche die Kante verbindet.

Der CLIP Algorithmus arbeitet in zwei Phasen. In der ersten Phase bestimmt das Verfahren die Kantenstärken und ermittelt Connected Components, welche ausschließlich durch Strong-Links verbunden sind. Handelt es sich bei einer Komponente um ein vollständiges Cluster, welches jeweils eine Beschreibung aus jeder Quelle enthält, dann wird dieses aus dem Graphen entfernt und in die Ausgabe übernommen. In Phase zwei ermittelt CLIP erneut Connected Components, welche diesmal ausschließlich durch Strong- und Normal-Links verbunden sind. Falls eine der Komponenten Clean-Source-Konsistent ist, wird sie als Cluster in die Ausgabe übernommen. Andernfalls ist eine weitere Verarbeitung erforderlich.

CLIP sortiert alle Kanten einer inkonsistenten Komponente (ausgenommen Weak-Links) anhand ihrer Priorität. Diese ergibt sich aus dem Kantengewicht (der Ähnlichkeitswert), der Kantenstärke und dem Kantengrad. In der CLIP-Konfiguration lässt sich die Gewichtung der drei Faktoren definieren. Jede Entitätsbeschreibung der Komponente wird zunächst als ein einzelnes *Singleton-Cluster* betrachtet. CLIP prüft nun für jede Kante, in absteigender Reihenfolge entsprechend der Kantenpriorität, ob sich die beiden durch die Kante verbunden Cluster zusammenführen lassen, ohne die Clean-Source-Konsistenz zu verletzen. Ist dies der Fall, werden die Cluster zusammengeführt, so dass am Ende jede Gruppe maximal eine Beschreibung aus der gleichen Quelle enthält. [37, S. 72 ff.]

3.2 Apache Flink

Apache Flink [3] ist ein Open-Source System für die Verarbeitung von Datenströmen (engl. Stream-Processing) und Stapeldaten (engl. Batch-Processing). Seine Architektur als verteiltes System erlaubt die Arbeit mit großen Datenmengen im Big-Data-Umfeld. Dabei wird die Rechenleistung „horizontal“ skaliert. Im Gegensatz zur „vertikalen“ Skalierung, bei der ein einzelner Computer mit immer leistungsstärkeren Hardware-Komponenten ausgerüstet wird, erhöht sich die Rechenleistung bei der horizontalen Skalierung durch die parallele Datenverarbeitung mehrerer verbundener Computer-Systeme. [34, S. 6 f.]

Apache Flink folgt der Philosophie, dass sich jegliche Datenverarbeitung auf einen fehlertoleranten, parallelen Datenfluss abbilden lässt. Dieser Datenfluss wird in Flink als sogenannter *Job-Graph* modelliert. Ein Job-Graph besteht aus einer Reihe beliebiger Aufgaben, die Datenströme als Eingabe konsumieren, verarbeiten und als Ausgabe an andere Aufgaben weiterreichen. Somit werden in Flink sämtliche Anwendungsfälle durch ein einheitliches, datenstrombasiertes Ausführungsmodell umgesetzt. [3, S. 28 ff.]

Flink besteht aus einer Reihe aufeinander aufbauender Komponenten in einem Schichtenmodell (vgl. Abb. 3.2.1). Der Kern des Systems ist die *Runtime-Schicht*. Sie führt in Form eines Job-Graphen modellierte Programme als verteilte Datenflüsse aus. Die Ausführung kann auf einem einzelnen Computer oder auf verteilten Systemen erfolgen. Die Job-Graphen werden von den Programmierschnittstellen (API, engl. *application programming interface*) der darüberliegenden Schicht erzeugt. Die *DataStream-API* dient der Generierung von Programmen für Datenströme. Mit der *DataSet-API* lassen sich Job-Graphen für Stapeldaten erzeugen. Flink abstrahiert

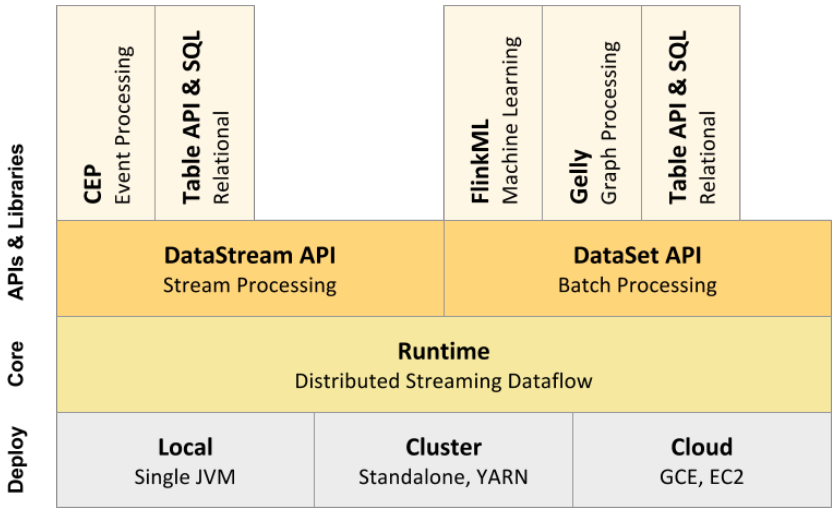


Abbildung 3.2.1: Flink Software-Stack [45]

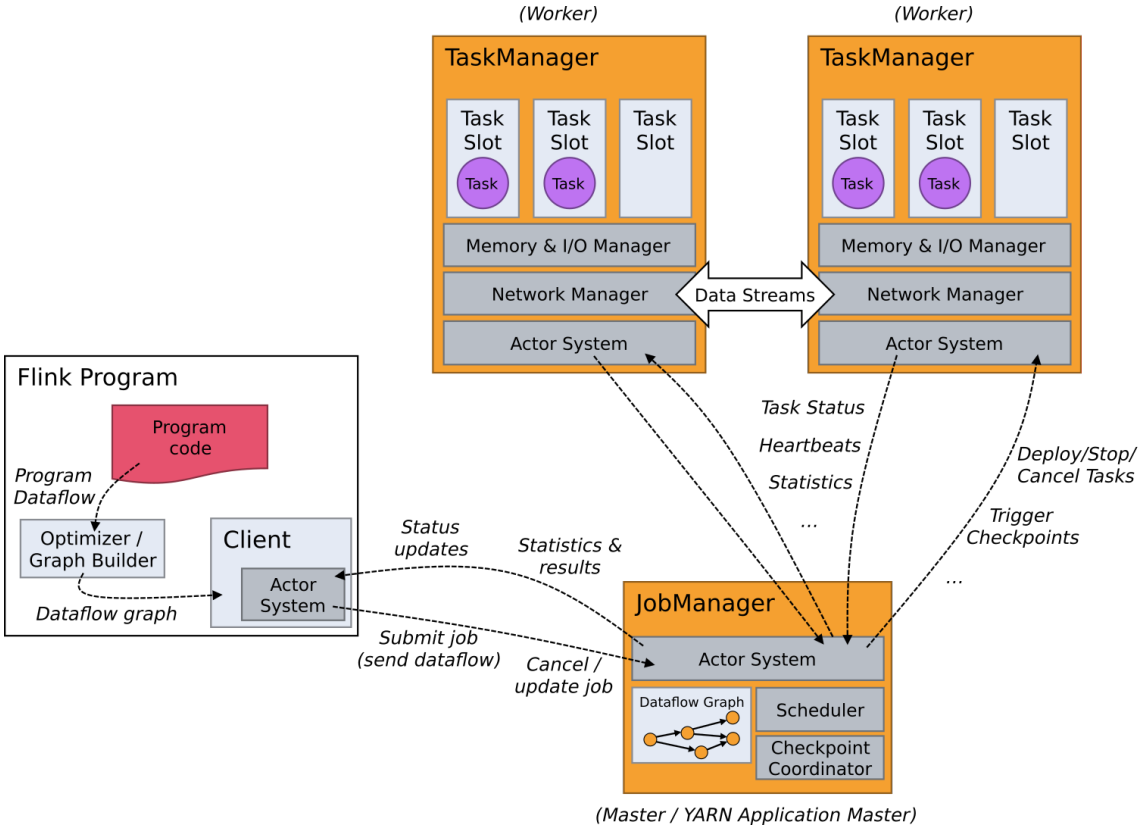


Abbildung 3.2.2: Flink Prozess-Modell [46]

die Stapelverarbeitung als einen Spezialfall von Streaming, bei dem der Datenstrom endlich ist und die Reihenfolge und Zeit der einzelnen Tupel keine Rolle spielt [3, S. 29]. Domänenspezifische Bibliotheken erleichtern die Entwicklung von Programmen mit der jeweiligen API. So lassen sich beispielsweise mit *Gelly* Flink-Programme für die Analyse von Graphen implementieren. [45]

Die Flink Runtime besteht aus zwei verschiedenen Typen von Prozessen, den Task-Managern und den Job-Managern. Sie sind auf Abbildung 3.2.2 in orange dargestellt. Die Prozesse lassen sich auf einem einzelnen Computer als Standalone-Cluster oder auf verteilten Computer-Clustern starten. Dabei ist es möglich, Frameworks wie *Apache Hadoop YARN* zur Ressourcenverwaltung einzusetzen. Der Client ist kein Teil der Flink Runtime. Er transformiert den Programmcode in einen Job-Graphen und sendet ihn an den Job-Manager. Der Job-Manager koordiniert die verteilte Ausführung des Datenflusses. Er verfolgt den Status und den Fortschritt jeder Aufgabe und weist den Task-Managern neue Aufgaben zu. Die eigentliche Datenverarbeitung findet in den Task-Managern statt. Sie verarbeiten eine oder mehrere Aufgaben parallel und berichten deren Status an den Job-Manager. Die ein- bzw. ausgehenden Datenströme der einzelnen Aufgaben werden direkt zwischen den Task-Managern ausgetauscht. [3, S. 29 f.][46]

3.2.1 Die DataSet-API

Die DataSet-API dient der verteilten Verarbeitung von Stapeldaten. Das Programmiermodell basiert auf Konzepten von Dean und Ghemawats (Google Inc.) *MapReduce* [10]. Zum besseren Verständnis der Datenverarbeitung in Flink wird zunächst das Basismodell MapReduce erläutert und anschließend auf Eigenheiten des Flink Programmiermodells eingegangen.

MapReduce ermöglicht die parallele Verarbeitung großer Datenmengen in einem horizontal skalierbaren Computer-System. Abbildung 3.2.3 zeigt einen beispielhaften MapReduce-Prozess, in dem die Anzahl verschiedener Datenträgertypen in einem Datensatz gezählt wird. Der Prozess besteht aus drei Phasen: *Map, Shuffle-and-Sort* und *Reduce*. Die Eingabedaten werden zu Beginn in sogenannte *Splits* partitioniert, welche auf der Abbildung lila dargestellt sind. In der Map-Phase erzeugt eine vom Benutzer definierbare Map-Funktion Schlüssel-Wert-Paare (orange) aus den Eingabedaten. Dies geschieht parallel auf verschiedenen Rechner-Knoten. Im Beispiel verarbeiten drei Rechner fünf Splits. Für das einfache Zähl-Beispiel bildet der Datenträgertyp den Schlüssel und der Wert beträgt eins. In der Shuffle-and-Sort-Phase werden die Schlüssel-Wert-Paare nach Schlüsseln sortiert und als Gruppe verschiedenen Rechner-Knoten zugewiesen. So erhält im Beispiel der erste Reduce-Knoten alle Schlüssel-Wert-Paare für CDs und Blu-rays, während der zweite alle Paare für DVDs und Videos empfängt. Die Reduce-Phase berechnet mittels einer benutzerdefinierbaren Reduce-Funktion die Ergebnisse des Prozesses. Im Beispiel summiert jeder Reduce-Knoten die Werte aller seiner Schlüssel-Gruppen. Somit ergibt sich in der Ausgabe die Anzahl jedes Datenträgertyps (grün dargestellt). [27][35, S. 5 ff.]

Das in [47] beschriebene Programmiermodell von Apache Flink bietet weitaus mehr Flexibilität als das einfache MapReduce. Als Werte lassen sich Tupel aus bis zu 25 Feldern beliebigen Datentyps oder einfache *Plain Old Java Objects* (POJO) verwenden. Jedes Tupel-Feld oder POJO-Attribut kann als Schlüssel dienen, ohne

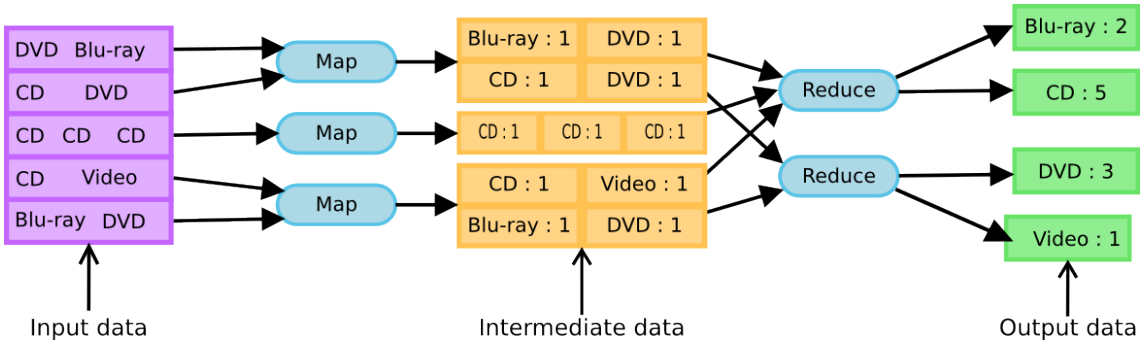


Abbildung 3.2.3: Beispiel eines MapReduce-Prozesses [27]

dass zuvor eine Map-Funktion erforderlich ist. *Map* hat in Flink eine völlig andere Bedeutung und steht für eine Eins-zu-eins-Transformation der Werte eines *DataSets*. Ein DataSet ist ein Datenstromelement, welches eine Menge gleichartiger Tupel oder POJOs enthält. Das Programmiermodell von Apache Flink bietet neben Map- und Reduce-Funktionen viele weitere mögliche Transformationen der Datenströme an. Die für das Verständnis dieser Arbeit relevanten Operationen werden anhand von [47] am Ende dieses Abschnitts einzeln erläutert. Die Transformationen lassen sich verketten und so als ein Datenfluss in einem Job-Graphen modellieren. Der Datenfluss kann verschiedene Datenquellen beinhalten, Zwischenergebnisse persistieren, sich verzweigen und wieder zusammengeführt werden. Die DataStream-API basiert im Grundsatz auf dem gleichen Konzept. Sie verfügt jedoch über weitere, für Datenströme relevante Funktionalitäten, wie beispielsweise Windowing. Da für diese Arbeit nur die DataSet-API relevant ist, wird auf die DataStream-API nicht weiter eingegangen.

Map: Mit einer Map-Transformation lässt sich jedes Element eines DataSets einzeln verarbeiten und auf einen anderen Wert abbilden. Das Ergebnis-DataSet enthält also die gleiche Anzahl von Elementen wie das Eingabe-DataSet.

FlatMap: Die FlatMap-Funktion ermöglicht eine M-zu-N-Abbildung eines DataSets. Jedes Element des Eingabe-DataSets kann auf keines, eines oder mehrere Elemente des Ergebnis-Sets abgebildet werden.

GroupBy: Die GroupBy-Funktion gruppiert ein DataSet nach gewissen Kriterien für die spätere Verarbeitung in einer anderen Transformation wie beispielsweise Reduce. Die Gruppierung kann über mehrere Tupel-Felder, POJO-Attribute oder durch eine benutzerdefinierte Funktion erfolgen.

Reduce: Reduce lässt sich auf einem gruppierten oder einem vollständigen DataSet (entspricht einer einzelnen Gruppe) ausführen. Jede Gruppe wird im Ergebnis auf ein einzelnes Element reduziert. Dies geht paarweise vonstatten. Die Funktion verarbeitet immer jeweils ein Elementenpaar des DataSets und reduziert es auf ein einzelnes Element, welches wiederum mit einem anderen verglichen und reduziert wird. Dieser Vorgang wiederholt sich so lange, bis nur noch ein einzelnes Element für jede Gruppe übrig bleibt.

GroupReduce: Im Gegensatz zu Reduce verarbeitet eine GroupReduce-Transformation eine komplette Gruppe auf einmal. Sie kann über alle Gruppenelemente iterieren und eine beliebige Anzahl von Ergebniselementen ausgeben.

Join: Join verbindet zwei DataSets zu einem einzelnen. Ähnlich wie bei GroupBy lässt sich der Schlüssel für das Zusammenführen der Elemente beider DataSets aus mehreren Tupel-Feldern, POJO-Attributen oder mit Hilfe einer speziellen Funktion bestimmen. Die verbundenen Ergebnis-Elemente können durch eine zusätzliche Funktion beliebig aus den zusammengeführten Elementen erzeugt werden. Flink unterstützt innere, äußere und Cross-Joins.

Delta-Iteration: Die Delta-Iteration ist eine Möglichkeit, um in Flink mehrere Datenfluss-Transformationen wiederholt in einem iterativen Prozess auszuführen. Die Operationen einer Iteration werden in einer sogenannten *Step-Funktion* gekapselt, so dass sich die Ergebnisse verschiedener Iterationen nicht vermischen. Ein WorkingSet bildet die Eingabe jeder Iteration und wird während der Step-Funktion für die nächste Iteration gefüllt. Das SolutionSet enthält die endgültigen Ergebnisse und kann schrittweise über mehrere Iterationen gefüllt werden. Der iterative Prozess endet, wenn das WorkingSet keine Elemente mehr enthält. Dies ermöglicht die Definition eines Vorgangs, bei dem ein Großteil des Ergebnisses in den ersten Iterationen ermittelt wird und somit spätere Iterationen nur noch auf einer geringen Teilmenge der Daten arbeiten müssen.

Die DataSet-API verfügt über eine Reihe weiterer Datenfluss-Transformationen, die auch für die prototypschen Implementierungen in Kapitel 5 Einsatz finden, jedoch in den kompakten Erläuterungen eingespart wurden. Dazu gehören verschiedene vordefinierte Aggregationen, Filter-Funktionen, Vereinigungen und eine andere Möglichkeit für iterative Prozesse.

3.2.2 Graphanalyse mit Gelly

Gelly ist eine Bibliothek zur Graphanalyse der Flink DataSet-API. Sie abstrahiert das Programmiermodell der DataSet-API und stellt High-Level-Funktionen für das Erstellen, Modifizieren und Transformieren von Graphen bereit. Mit Hilfe iterativer Message-Passing-Modelle lassen sich Algorithmen zur Verarbeitung von Graphen entwerfen. Gelly stellt außerdem eine Reihe fertiger Algorithmen zur Graphanalyse bereit, wie beispielsweise zum Identifizieren von Teilgraphen (Connected Components). Ein Graph besteht in Gelly aus zwei DataSets, eines für die Knoten und

eines für die Kanten. Knoten sind Zwei-Feld-Tupel aus ID und Knotenwert. Eine Kante ist ein Tupel aus drei Feldern, jeweils eines für jede ID der beiden Knoten, die sie verbindet und eines für den Kantenwert. Knoten- und Kantenwert können von einem beliebigen Datentyp oder auch ein POJO sein. Mit dieser Art der Repräsentation kann Gelly die Funktionen der DataSet-API einsetzen, um einen Graphen zu verarbeiten. [48]

Gelly bietet drei verschiedene Modelle für die iterative Verarbeitung von Graphen: *Vertex-Centric*, *Scatter-Gather* und *Gather-Sum-Apply*. Das erste ist das allgemeinste der drei Modelle und bietet die meiste Flexibilität für den Algorithmenentwurf. Mit ihm lässt sich ein Algorithmus aus der Sichtweise eines Knotens formulieren. Die anderen beiden Modelle sind für diese Arbeit nicht relevant.

Abbildung 3.2.4 zeigt, wie Flink einen iterativen Prozess des knotenzentrierten Modells ausführt. Die kreisförmig dargestellten Knoten tauschen Nachrichten auf den Kanten des Graphen aus. Jede Iteration wird als *Superstep* bezeichnet und ist strikt von der nächsten abgetrennt. Innerhalb eines Supersteps werden sämtliche Knoten parallel verarbeitet. Die gepunkteten Rechtecke zeigen die parallel ausgeführten Einheiten. Jeder Knoten liest seine empfangenen Nachrichten, die ihm seine Nachbarn in der vorangegangenen Iteration zugestellt haben, manipuliert gegebenenfalls seinen Knotenwert und sendet wiederum selbst Nachrichten an seine Nachbarknoten. Flink beginnt den nächsten Superstep, wenn die Verarbeitung jedes Knotens im vorangegangenen Superstep abgeschlossen ist. Diese strikte Trennung garantiert, dass sich die Nachrichten verschiedener Iterationen nicht vermischen.

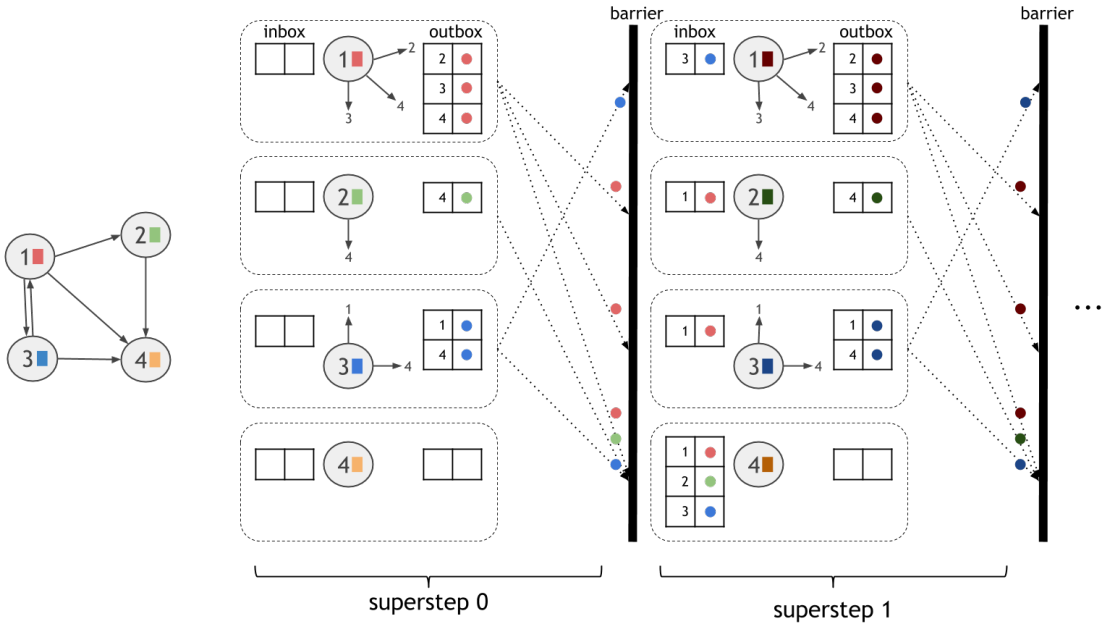


Abbildung 3.2.4: Knotenzentriertes Iterationsmodell in Gelly [49]

Für einen knotenzentrierten Message-Passing-Algorithmus genügt die Definition einer einzelnen Funktion für die Aktion eines Knotens innerhalb eines Supersteps. Mit Hilfe von Aggregatoren können globale Informationen über alle Knoten gesammelt und in der nächsten Iteration abgerufen werden. Eine optionale *MessageCombiner*-Funktion kann die empfangenen Nachrichten eines Knotens kombinieren und diesem einen einzelnen Wert als Eingangsnachricht übermitteln. Dies erspart Kosten im zugrundeliegenden DataSet-API Datenfluss. Ein Knoten muss nicht jedem seiner Nachbarn eine Nachricht senden. Flink führt die Funktion eines Knotens nur dann aus, wenn dieser mindestens eine Nachricht empfangen hat. Der iterative Prozess endet, wenn kein Knoten mehr eine Nachricht empfängt oder eine definierbare Anzahl von Iterationen erreicht ist. Das Ergebnis eines knotenzentrierten Algorithmus ist ein neuer Graph mit aktualisierten Knotenwerten. [49]

3.3 FAMER

FAMER ist ein an der Universität Leipzig entwickeltes, skalierbares ER-System. Es basiert auf Apache Flink und ermöglicht daher eine verteilte Ausführung auf einem Computer-Cluster. Dadurch ist es in der Lage, große Datenmengen im Big-Data-Umfeld parallel zu verarbeiten. FAMER unterstützt bisher MS-Clean-ER und MS-Dirty-ER (vgl. Abschnitt 2.1) für eine beliebige Anzahl von Datenquellen. Im Rahmen dieser Masterarbeit soll es um ein Clustering-Verfahren für MSCD-ER erweitert werden, sodass es sämtliche Varianten von Multi-Source-ER unterstützt.

Abbildung 3.3.1 zeigt den Arbeitsablauf des ER-Prozesses in FAMER. Er richtet sich nach dem in Abschnitt 3.1 beschriebenen grundlegenden ER-Prozess. Die Eingabe besteht aus einer beliebigen Anzahl von Datenquellen mit Entitätsbeschreibungen. Die Ausgabe bildet eine Menge von disjunkten Clustern, in der versucht wird, alle Beschreibungen der gleichen Entität dem gleichen Cluster zuzuordnen. Der Vorgang besteht aus zwei Hauptkomponenten, dem *Similarity-Graph-Generator* und dem *Entity-Clustering*, wobei erstere mit dem Similarity Graph die Eingabe für letztere erzeugt. Die beiden Komponenten sind getrennt dargestellt, weil sie unabhängig voneinander ausgeführt werden können. FAMER erlaubt es, den Similarity-Graph als Zwischenergebnis zu persistieren, um so verschiedene Clustering-Verfahren testen zu können, ohne jedes Mal den gesamten ER-Prozess durchlaufen zu müssen.

Im Similarity Graph Generator sind die Schritte Blocking und Block-Processing des grundlegenden ER-Prozesses unter dem Begriff Blocking zusammengefasst. FAMER unterstützt derzeit Standard-Blocking und Sorted-Neighborhood-Blocking sowohl als Single- als auch als Multi-Pass-Variante für mehrere Blocking-Keys (vgl.

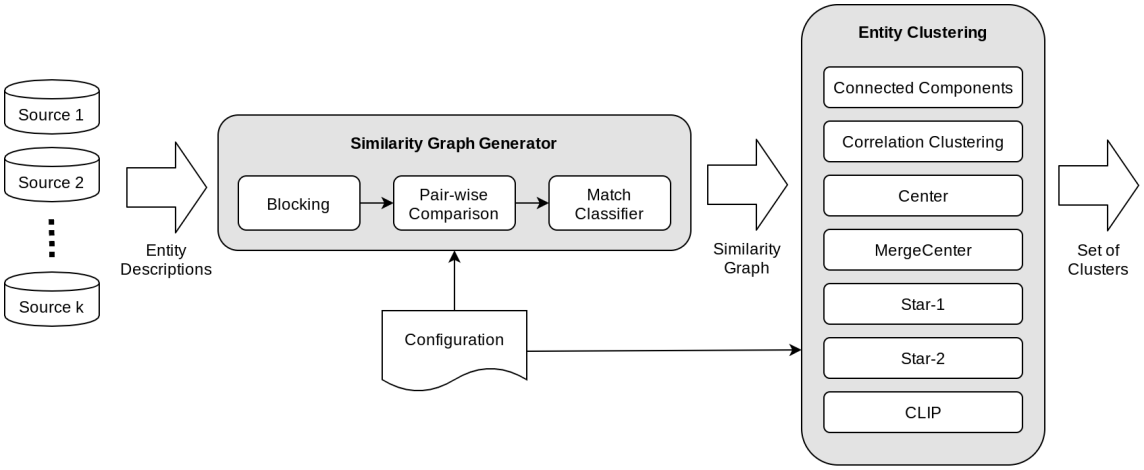


Abbildung 3.3.1: *Übersicht über den Arbeitsablauf von FAMER für Multi-Source-ER.* Die Grafik wurde aus [37] entnommen und für den derzeitigen Stand von FAMER angepasst.

Abschnitt 3.1.1). Durch das in [24] vorgestellte *Block-Split*-Verfahren ermöglicht es außerdem, große Blöcke auf verschiedenen Task-Managern parallel zu verarbeiten. [37, S. 64]

Die Ähnlichkeits- und Match-Funktion des Entity-Matchings werden in FAMER getrennt als *Pair-wise Comparison* und *Match-Classifier* aufgeführt. Das System unterstützt zwölf verschiedene Methoden zur Berechnung der Ähnlichkeit von Zeichenketten. Des Weiteren ermöglicht es die Berechnung numerischer Ähnlichkeiten und die Bestimmung des Abstands zweier geographischer Koordinaten. Ähnlichkeits- und Match-Funktion lassen sich manuell in einer Konfigurationsdatei definieren. In Zukunft soll das System eine Match-Klassifizierung anhand von Trainingsdaten mittels eines maschinellen Lernverfahrens unterstützen [37, S. 65]. FAMER unterstützt derzeit sieben verschiedene Clustering-Verfahren (vgl. Abb. 3.3.1), die sich über die Konfigurationsdatei auswählen und parametrisieren lassen.

Zuletzt wurde FAMER für die Unterstützung inkrementeller ER erweitert [39]. Damit ist es möglich, neue Entitätsbeschreibungen in ein bestehendes Clustering zu integrieren, ohne den ER-Prozess für den gesamten Datensatz wiederholen zu müssen. Außerdem unterstützt es das „Reparieren“ des Clustering-Ergebnisses, indem für einen definierbaren Anteil der Beschreibungen ein *Re-Clustering* ausgeführt wird. Dies ist nötig, wenn so viele neue Entitätsbeschreibungen hinzugekommen sind, dass mit diesen zusätzlichen Informationen initial eine genauere Gruppierung möglich gewesen wäre.

3.4 Affinity Propagation

Frey und Dueck stellen in [12] das exemplarbasierte Clustering-Verfahren *Affinity Propagation* (AP) vor. Da sich AP besonders gut für Erweiterungen eignet (vgl. Abschnitt 3.4.3), soll es im Rahmen dieser Masterarbeit um die Funktion des MSCD-Clusterings ergänzt werden. In AP ist ein „Exemplar“ der Repräsentant eines Clusters, also jener Datenpunkt, der das Cluster am besten nach außen vertritt. Im Zusammenhang mit AP wird im Folgenden von Datenpunkten anstelle von Entitätsbeschreibungen gesprochen, da dies dem Vokabular der Originalpublikation entspricht und der Algorithmus, außerhalb des Kontexts von Entity Resolution, als allgemeines Clustering-Verfahren entwickelt wurde. Ein Exemplar bildet das Zentrum eines Clusters, dem die anderen Datenpunkte des Clusters zugeordnet sind. Ziel von AP ist es, eine Menge von Exemplaren und Exemplarzuordnungen der Datenpunkte zu finden, so dass die Summe der Ähnlichkeiten zwischen den Exemplaren und den ihnen zugeordneten Punkten maximal ist. Dieses Ziel lässt sich als eine Energiefunktion definieren, die es zu maximieren gilt. Es handelt sich beim Clustering mit AP also um ein Optimierungsproblem. Der Algorithmus bildet sternförmige Cluster. Alle gemeinsam gruppierten Datenpunkte besitzen im Similarity-Graph eine Kante zu dem Exemplar des Clusters.

AP basiert auf zwei Konzepten: *Faktorgraphen* und dem *Max-Sum-Algorithmus*. Die Energiefunktion lässt sich mit Hilfe einer Faktorisierung als ein Faktorgraph darstellen. Durch Anwendung des Max-Sum-Algorithmus lässt sich die Energiefunktion auf dem Faktorgraph maximieren. In den folgenden Abschnitten werden die beiden Konzepte vorgestellt und anschließend erläutert, wie sich AP daraus ableiten lässt.

3.4.1 Faktorgraphen

Der Faktorgraph ist eine generalisierte Form des Tanner-Graphen [44], der von Forney et al. entwickelt wurde [14]. Das Konzept des Faktorgraphen bietet ein Framework für die Entwicklung von Algorithmen für Optimierungsprobleme. Diese Algorithmen sind Abwandlungen des *Belief Propagation* Algorithmus von Pearl [32]. Eine dieser Abwandlungen ist der Max-Sum-Algorithmus, auf dem AP basiert und der im nächsten Abschnitt erläutert wird.

Eine komplexe Energiefunktion, die von einer großen Menge von Variablen abhängt, lässt sich durch eine Faktorisierung in ein Produkt sogenannter „lokaler“ Funktionen zerlegen. Jede lokale Funktion hängt jeweils nur von einer Teilmenge der Variablen ab und lässt sich dadurch einfacher definieren und interpretieren. Formel 3.4.1 zeigt eine Faktorisierung an einem einfachen Beispiel von fünf Variablen

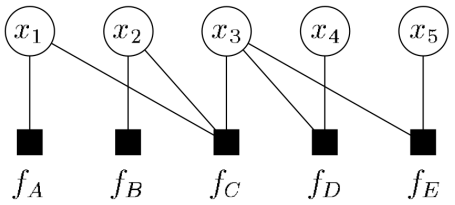


Abbildung 3.4.1: Beispielhafter Faktorgraph für die Faktorisierung in Formel 3.4.1 [25, S. 500]

$x_1, .., x_5$ und fünf lokalen Funktionen $f_A, .., f_E$, die keine nähere Bedeutung besitzen [25, S. 499].

$$e(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) \cdot f_B(x_2) \cdot f_C(x_1, x_2, x_3) \cdot f_D(x_3, x_4) \cdot f_E(x_3, x_5) \quad (3.4.1)$$

Ein Faktorgraph ist die Visualisierung einer solchen Faktorisierung in Form eines bipartiten Graphs. Dieser besteht aus zwei Knotenmengen, einer Menge für die Variablen und einer für die lokalen Funktionen (auch Faktorknoten genannt). Ein Faktorknoten und ein Variablenknoten werden durch eine ungerichtete Kante verbunden, wenn die lokale Funktion des Faktorknotens von der Variablen abhängig ist. Abbildung 3.4.1 zeigt den Faktorgraph für die Faktorisierung aus Formel 3.4.1. Variablenknoten sind als Kreise dargestellt, Faktorknoten in Form von Quadraten.

3.4.2 Der Max-Sum-Algorithmus

Der Max-Sum-Algorithmus ist ein von Pearls *Belief Propagation* [32] abgeleiteter Message-Passing-Algorithmus. Er dient dazu, die Konfiguration der Variablen einer Zielfunktion zu finden, die diese maximieren. In Kombination mit einem Faktorgraph dient der Algorithmus der sogenannten *Constraint-Optimierung*. Dabei wird eine Zielfunktion bezüglich ihrer Variablen maximiert, wobei gewisse Einschränkungen (engl. constraints) zu berücksichtigen sind. Diese Eigenschaft machen sich Frey und Dueck bei der Entwicklung des AP-Algorithmus in [12] zu Nutze, worauf im nachfolgenden Abschnitt näher eingegangen wird.

Kschischang et al. übertragen das ursprünglich für bayessche Netze entworfene Belief Propagation (auch *Sum-Product-Algorithmus* genannt) in [25] auf die Anwendung in einem Faktorgraph. An diese Arbeit angelehnt erfolgt nun eine Beschreibung des Max-Sum-Algorithmus für Faktorgraphen, die sich auf die für AP wesentlichen Schritte beschränkt. Die Gleichungen des Algorithmus wurden der Formulierung von Zabaras in [51, S. 22 f.] entnommen.

Die Faktor- und Variablenknoten tauschen Nachrichten auf den Kanten des Faktorgraphen aus. Jede Nachricht, die ein Knoten auf einer Kante sendet, ist dabei

eine effektive Zusammenfassung aller Nachrichten, die er auf allen anderen Kanten erhalten hat. Ein Faktorknoten teilt einem Variablenknoten mit, welchen Wert die Variable einnehmen sollte. Diese Einschätzung basiert auf der (Constraint-) Funktion des Faktorknotens und den Werten aller anderen Variablenknoten, die mit ihm benachbart sind. Eine Nachbarschaft bedeutet in diesem Kontext, dass eine Kante zwischen den Knoten vorhanden ist. Ein Variablenknoten berechnet und sendet seinen Wert an einen benachbarten Faktorknoten, basierend auf den Funktionsergebnissen aller anderen benachbarten Faktorknoten. Diese Berechnung enthält indirekt die Werte der anderen Variablen, da diese von den Faktorknoten bei der Ermittlung ihres Funktionsergebnisses berücksichtigt wurden. In einem zyklenfreien Graph stößt der Nachrichtenempfang an einer Kante das Senden von Nachrichten auf allen anderen Kanten eines Knotens an. Das Verfahren terminiert, wenn keine Nachricht mehr gesendet werden kann. Dabei findet der Max-Sum-Algorithmus eine nachweislich exakte Lösung [13, S. 9]. In zyklischen Graphen arbeitet der Algorithmus in einem iterativen Verfahren und findet eine approximative Lösung. Dabei ist eine Iteration dann abgeschlossen, wenn auf jeder Kante in jeder Richtung eine Nachricht ausgetauscht wurde.

Abbildung 3.4.2 zeigt die Nachrichten, die ein Variablenknoten x und ein Faktorknoten f im Max-Sum-Algorithmus auf einem Faktorgraphen austauschen. Die Knotenmenge $n(x) \setminus \{f\}$ enthält alle benachbarten Knoten n von x, ausgenommen von f, da x eine Nachricht zu f sendet. Auf die gleiche Weise enthält die Menge $n(f) \setminus \{x\}$ alle Nachbarn von f, ausgenommen von x. Die Nachricht $\mu_{x \to f}(x)$ von der Variablen x zum Faktor f (vgl. Formel 3.4.2) ergibt sich aus der Summe aller in x eingegangenen Nachrichten der anderen benachbarten Faktorknoten $h_1, ..., h_l$. Die Nachricht $\mu_{f \to x}(x)$ von f zu x (vgl. Formel 3.4.3) ergibt sich aus der Maximierung

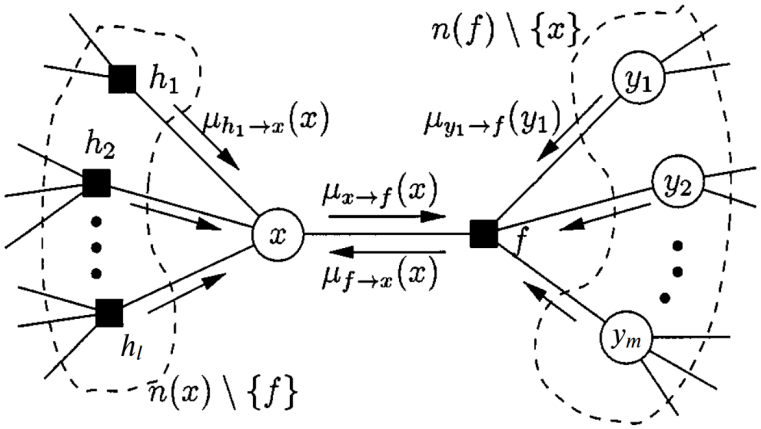


Abbildung 3.4.2: Auszug eines Faktorgraphen zur Illustration des Nachrichtenaustauschs im Max-Sum-Algorithmus [25, S. 502]

der Summe des Logarithmus der Funktion des Faktors f und aller in f eingegangenen Nachrichten der Variablenknoten $y_1, ..., y_m$, über alle möglichen Konfigurationen der Variablen in $n(f) \setminus \{x\}$.

Variablenknoten zu Faktorknoten:

$$\mu_{x \to f}(x) = \sum_{h_i \in n(x) \setminus \{f\}} \mu_{h_i \to x}(x) \tag{3.4.2}$$

Faktorknoten zu Variablenknoten:

$$\mu_{f \to x}(x) = \max_{n(f) \setminus \{x\}} \left(lnf(x, y_1, ..., y_m) + \sum_{y_i \in n(f) \setminus \{x\}} \mu_{y_i \to f}(y_i) \right) \tag{3.4.3}$$

Initiale Nachrichten:

$$\mu_{x \to f}(x) = 0 \tag{3.4.4}$$

$$\mu_{f \to x}(x) = lnf(x) \tag{3.4.5}$$

Für einen zyklischen Faktorgraph wie in AP wird in der ersten Iteration des Max-Sum-Algorithmus davon ausgegangen, dass eine initiale Nachricht, entsprechend der Formeln 3.4.4 und 3.4.5, an jeder Kante eines jeden Knotens eingegangen ist. Dadurch ist jeder Knoten in der Lage, an jeder Kante eine Nachricht zu versenden, da er die initialen Nachrichten zur Berechnung der zu versendenden Werte verwenden kann. Spätere Iterationen verwenden die eingegangenen Nachrichten der vorherigen Iteration zur Berechnung der ausgehenden Nachrichten. Der Max-Sum-Algorithmus wiederholt diesen Nachrichtenaustausch so lange, bis ein Abbruchkriterium oder eine definierte Anzahl von Iterationen erreicht wurde.

Belegung einer Variable, welche die Zielfunktion maximiert:

$$x_i^{max} = \arg\max_{x_i} \left(\sum_{f_i \in n(x)} \mu_{f_i \to x}(x) \right) \tag{3.4.6}$$

Entsprechend Formel 3.4.6 lässt sich nach jeder Iteration an jedem Variablenknoten x_i die aktuell ermittelte Variablenbelegung bestimmen, welche die Energiefunktion maximiert. Dazu wird aus allen möglichen Werten von x_i jener gewählt, welcher die Summe aller eingehenden Nachrichten zu dem Variablenknoten maximiert.

3.4.3 Affinity Propagation als Max-Sum-Instanz auf einem Faktorgraphen

AP ist eine Instanz des Max-Sum-Algorithmus auf einem Faktorgraphen. Frey und Dueck stellen in [13, S. 10 ff.] zwei verschiedene Topologien von Faktorgraphen vor. AP wurde ursprünglich mit Hilfe der ersten Topologie entwickelt. Sie verwendet eine diskrete Variable für jeden Datenpunkt, die jeweils die ID des Exemplars enthält, welches der jeweilige Datenpunkt gewählt hat. Die Herleitung von AP mit Hilfe dieses Modells ist sehr komplex. Givoni und Frey stellen deshalb in [18] eine Herleitung auf Basis der zweiten Topologie bereit. Diese verwendet N^2 binäre Variablen für N Datenpunkte, um deren Exemplarentscheidungen zu speichern. Abbildung 3.4.3 zeigt den Faktorgraphen für dieses Binärvariablenmodell. Dort sind die Variablen in Form einer quadratischen Matrix $\mathbb{B}$ angeordnet, in der die Zeilen durch den Index i und die Spalten durch den Index j nummeriert sind. Die Binärvariable b_{ij} enthält den Wert eins, wenn Datenpunkt j ein Exemplar und Punkt i diesem zugeordnet ist. Andernfalls enthält die Variable den Wert null.

Das Binärvariablenmodell eignet sich besonders gut, um Erweiterungen und Anpassungen an AP vorzunehmen, da sich die Nachrichtenwerte damit einfacher aus den Formeln des Max-Sum-Algorithmus ableiten lassen [16, S. 50]. Einige der Nachrichtenformeln von Max-Sum erfordern, dass alle möglichen Variablenbelegungen berücksichtigt werden. Dies lässt sich mit Binärvariablen einfacher als mit diskreten Variablen erreichen. Aus diesem Grund beschränkt sich die vorliegende Masterarbeit im Folgenden auf das Binärvariablenmodell.

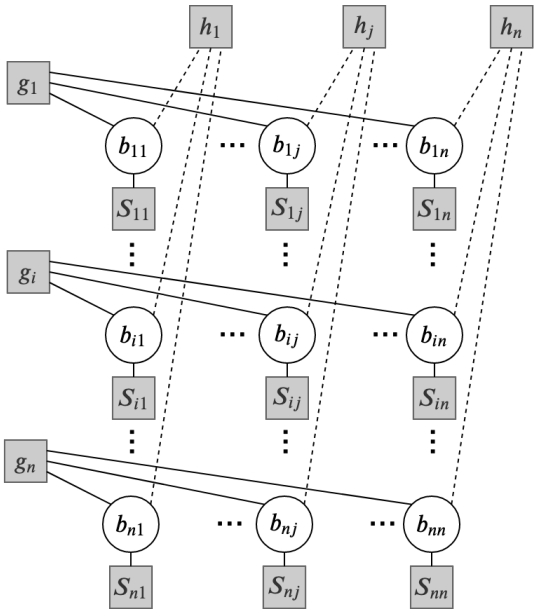


Abbildung 3.4.3: Faktorgraph von AP für das Binärvariablenmodell [23, S. 3]

i \ j	**0**	**1**	**2**	**3**	**4**
0	**1**	0	0	0	0
1	0	0	**1**	0	0
2	0	0	**1**	0	0
3	0	0	0	**1**	0
4	0	0	**1**	0	0

Tabelle 3.4.1: Binärmatrix $\mathbb{B}$ für das Clustering in Abb. 3.4.4

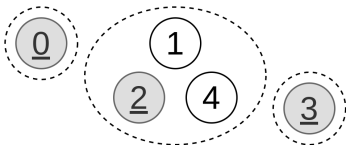


Abbildung 3.4.4: Beispielhaftes exemplarbasiertes Clustering

Abbildung 3.4.4 zeigt ein Beispiel für ein exemplarbasiertes Clustering von fünf Datenpunkten, welches in Tabelle 3.4.1 in Form einer Binärmatrix dargestellt ist. Es entspricht dem Einführungsbeispiel aus Abbildung 2.2.2a. Exemplare sind in der Abbildung grau hinterlegt und unterstrichen dargestellt. An den Zeilen der Binärmatrix lässt sich, wie oben beschrieben, die Exemplarzuordnung ablesen. Die Spalten der Matrix zeigen die Cluster an. Alle Datenpunkte, bei denen in einer gemeinsamen Spalte der Wert eins vorkommt, befinden sich im gleichen Cluster. Die Diagonale gibt Auskunft über die Exemplare des Clusterings. Jeder Datenpunkt, dessen Diagonalwert eins ist, repräsentiert ein Exemplar.

Formel 3.4.7 zeigt die Energiefunktion von AP für das Binärvariablenmodell. Die Formulierung entstammt [23, S. 2] und ist eine leichter zu erfassende Variante der Formulierung von Givoni und Frey in [17]. $\mathbb{B}(i,:)$ steht für alle Binärvariablen in Zeile i und $\mathbb{B}(:,j)$ entsprechend für alle Variablen in Spalte j. Im Max-Sum-Algorithmus wird eine Summe anstelle eines Produkts verwendet, um die einzelnen Komponenten der Energiefunktion zu verbinden. Dennoch bleibt die allgemeine Bezeichnung „Faktorgraph“ für die graphische Darstellung der Funktion bestehen.

$$E(\mathbb{B}) = \sum_{ij} s_{ij} b_{ij} + \sum_{i} g_i(\mathbb{B}(i,:)) + \sum_{j} h_j(\mathbb{B}(:,j)) \tag{3.4.7}$$

$$g_i(\mathbb{B}(i,:)) = \begin{cases} 0 & \text{wenn } \sum_j b_{ij} = 1 \\ -\infty & \text{andernfalls} \end{cases} \tag{3.4.8}$$

$$h_j(\mathbb{B}(:,j)) = \begin{cases} 0 & \text{wenn } b_{jj} = \max_i b_{ij} \\ -\infty & \text{andernfalls} \end{cases} \tag{3.4.9}$$

Die Faktorknoten g_i und h_i (vgl. Abb. 3.4.3) stellen ein valides Clustering sicher und werden deshalb als *Consistency-Constraints* bezeichnet. Jeder Zeile i von $\mathbb{B}$ wird jeweils ein Faktorknoten g_i hinzugefügt und jeder Spalte j ein Faktorknoten h_j. Die

Knoten $g_1, ..., g_n$ repräsentieren das sogenannte *1-of-N Constraint*. Sie erzwingen entsprechend Formel 3.4.8, dass jeder Datenpunkt i exakt einem Exemplar zugeordnet ist. In jeder Zeile der Binärmatrix $\mathbb{B}$ muss also exakt eine Eins vorkommen. Ist die Anforderung nicht erfüllt, wird ein Wert von $-\infty$ auf die Energiefunktion addiert, so dass diese bei der fehlerhaften Variablenkonfiguration nicht maximal sein kann. Auf die gleiche Weise stellt das *Exemplar-Consistency-Constraint* durch die Knoten $h_1, ..., h_n$ entsprechend Formel 3.4.9 sicher, dass ein Datenpunkt sich selbst auch als Exemplar wählt, wenn er durch einen anderen Punkt gewählt wurde. Kommt also in einer Spalte von $\mathbb{B}$ eine Eins vor, dann muss der Diagonalwert der Spalte auch eins sein. Dies garantiert, dass sich die Exemplare von der Diagonale ablesen lassen. Jede Binärvariable b_{ij} ist außerdem mit einem Faktorknoten S_{ij} verbunden. Dieser lässt die Information über die Ähnlichkeit zwischen den Punkten i und j in die Energiefunktion einfließen.

AP löst das Optimierungsproblem der Energiefunktion aus Formel 3.4.7 und findet eine optimale Belegung der Variablen in $\mathbb{B}$. In dieser Variablenbelegung ist die Summe jener Ähnlichkeiten S_{ij} maximal, welche mit Binärvariablen b_{ij} mit einem Wert von eins verbunden sind. Dies entspricht einer optimalen Auswahl von Exemplaren und Zuordnung der übrigen Datenpunkten zu diesen Exemplaren. Dabei garantieren die Consistency-Constraints ein valides Clustering. Eine exakte Maximierung der Energiefunktion ist rechnerisch nicht in polynomieller Zeit möglich, da es sich dabei um einen Spezialfall des NP-schweren k-Median-Problems handelt [12, S. 975]. AP findet eine Näherungslösung durch den Austausch der auf Abbildung 3.4.5 dargestellten Nachrichten zwischen den Knoten des Faktorgraphens. Die Formeln 3.4.10 - 3.4.13 sind aus dem Max-Sum-Algorithmus und der Energiefunktion in Formel 3.4.7 abgeleitet. Givoni [16, S. 54 ff.] nimmt eine Ausführliche Herleitung der einzelnen Nachrichtenformeln für das Binärvariablenmodell vor und interpretiert ihre Bedeutung.

AP verfügt über zwei Parameter, die das Ergebnis des Clusterings beeinflussen. Der erste Parameter ist die sogenannte *Preference*. Sie beeinflusst, wie wahrscheinlich ein Datenpunkt zu einem Exemplar wird. Je größer die Preference, desto eher bildet ein Datenpunkt das Zentrum eines Clusters. Der Parameter lässt sich im Similarity-Graph, über die Ähnlichkeit eines Knotens zu sich selbst, individuell für jeden Datenpunkt einstellen. Üblicherweise ist die Eignung der einzelnen Punkte, ein Exemplar zu bilden, nicht bekannt. Daher wird ein gemeinsamer Preference-Wert für alle Datenpunkte verwendet. Der Parameter beeinflusst die Anzahl der resultierenden Cluster von AP. Eine geringe gemeinsame Preference führt dabei zu einer kleinen Anzahl von großen Clustern. Entsprechend ergibt sich durch eine große

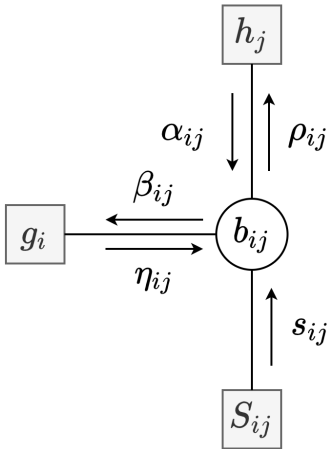


Abbildung 3.4.5: Nachrichtenaustausch zwischen den Variablen- und Faktorknoten in AP [23, S. 3]

$$\beta_{ij} = s_{ij} + \alpha_{ij} \tag{3.4.10}$$

$$\rho_{ij} = s_{ij} + \eta_{ij} \tag{3.4.11}$$

$$\eta_{ij} = -\max_{k \neq j} \beta_{ik} \tag{3.4.12}$$

$$\alpha_{ij} = \begin{cases} \sum_{k \neq j} \max(0, \rho_{kj}) & i = j \\ \min[0, \rho_{jj} + \sum_{k \notin \{i,j\}} \max(0, \rho_{kj})] & i \neq j \end{cases} \tag{3.4.13}$$

gemeinsame Preference eine hohe Anzahl kleiner Cluster. Frey und Dueck empfehlen den Median aller Ähnlichkeiten als gemeinsame Preference für eine moderate Cluster-Anzahl und das Minimum der Ähnlichkeiten für eine geringe Menge von Clustern [12, S. 972].

Der zweite Parameter von AP ist der Dämpfungsfaktor λ. Zu dessen Verständnis muss zunächst ein Problem von AP erläutert werden. Wenn mehrere mögliche Variablenbelegungen die Energiefunktion ähnlich gut maximieren, dann kommt es zu einer Oszillation zwischen den verschiedenen Lösungen. Das iterative Message-Passing-Verfahren konvergiert nicht. Tabelle 3.4.2 zeigt das anhand eines einfachen Beispiels. Der Similarity-Graph wird durch die Similarity-Matrix $\mathbb{S}$ abgebildet (vgl. Tab. 3.4.2a). Dort ist eine symmetrische Ähnlichkeit zwischen den Datenpunkten null und eins zu erkennen, die ein gemeinsames Cluster bilden. Da beide Punkte gleich gut als Exemplar geeignet sind, convergiert AP nicht für eine eindeutige Lösung. Stattdessen bildet, in abwechselnden Iterationen, jeweils einer der beiden Punkte das Zentrum des Clusters (vgl. Tab. 3.4.2b bis 3.4.2e). Zur Vermeidung solcher Oszillationen schlagen Frey und Dueck vor, ein geringes Rauschen auf die Ähnlichkeitswerte zu legen und eine Dämpfung der Nachrichtenwerte beim Message-Passing durchzuführen [12, S. 975]. Gedämpft werden die Nachrichten α und ρ [13, S. 2]. Jede Nachricht besteht zu einem Anteil von λ aus ihrem Wert der vergangenen

s_{ij}	**0**	**1**	**2**
0	0,8	**0,9**	0,1
1	**0,9**	0,8	0,1
2	0,1	0,1	0,8

(a) Similarity-Matrix $\mathbb{S}$

1	0	0
1	0	0
0	0	1

(b) Binärmatrix $\mathbb{B}$ bei Iteration t

0	**1**	0
0	**1**	0
0	0	1

(c) $\mathbb{B}$ bei Iteration t+1

1	0	0
1	0	0
0	0	1

(d) $\mathbb{B}$ bei Iteration t+2

0	**1**	0
0	**1**	0
0	0	1

(e) $\mathbb{B}$ bei Iteration t+3

Tabelle 3.4.2: Beispiel eines oszillierenden AP-Prozesses

Iteration und zu einem Teil von $1 - \lambda$ aus dem für die aktuelle Iteration errechneten Wert. Treten dennoch Oszillationen auf, ist das Rauschen zu erhöhen bzw. der Dämpfungsfaktor zu vergrößern.

Im Allgemeinen hat AP eine rechnerische Komplexität von $\mathcal{O}(N^2T)$ [28, S. 895], wobei N für die Anzahl der Datenpunkte und T für die zur Lösungsfindung benötigten Iterationen steht. Dies trifft zu, wenn die paarweisen Ähnlichkeiten von allen Datenpunktpaaren bekannt sind. Eine Similarity-Matrix, in der nahezu alle Elemente größer als null sind, wird als dichtbesetzte Matrix (engl. *dense matrix*) bezeichnet. In vielen Anwendungsfällen ist allerdings eine dünnbesetzte Similarity-Matrix (engl. *sparse matrix*) gegeben. So erzeugt beispielsweise das Entity-Matching in ER einen Similarity-Graph, dessen Kanten durch die Match-Funktion gefiltert werden (vgl. Abschnitt 3.1.3). Dieser Graph lässt sich auf eine dünnbesetzte Similarity-Matrix abbilden, in der die Zellen s_{ij} nur dann einen Wert enthalten, wenn eine Kante zwischen den Knoten i und j im Similarity-Graphen existiert. Dueck beschreibt in [11, S. 48], wie AP diesen Umstand ausnutzen kann, um den Rechenaufwand zu reduzieren. Ein Datenpunkt kann einem anderen nicht als Exemplar zugeordnet werden, wenn die Ähnlichkeit zwischen beiden Punkten unbekannt ist. Deshalb können die Binärvariablen b_{ij}, ihre Kanten und eventuell auch einige Faktorknoten aus dem Faktorgraph entfernt werden, wenn keine Ähnlichkeit zwischen i und j existiert. Dies verringert die Anzahl der Nachrichten, die berechnet werden müssen.

3.5 Adaptive Affinity Propagation

Im vorangegangenen Abschnitt wurde der Clustering-Algorithmus AP von Frey und Dueck vorgestellt. Das Verfahren hat zwei Schwachstellen. Zum einen ist die Qualität der Lösung sehr stark vom Preference-Parameter p abhängig, da dieser großen Einfluss auf Anzahl und Umfang der resultierenden Cluster hat. Zum anderen hat der Algorithmus Probleme, für eine bestimmte Lösung zu konvergieren, wenn verschiedene Variablenbelegungen die Energiefunktion ähnlich gut optimieren (vgl. Abschnitt 3.4.3). AP müsste mehrfach mit erhöhtem Dämpfungsfaktor λ oder mit ver-

stärktem Rauschen ausgeführt werden, um eine konvergierende Lösung zu finden. Wang et al. Wang et al. stellen in [50] die Erweiterung *Adaptive AP* vor, welche versucht, diese beiden Schwachstellen von AP zu beheben.

Um das Problem der Oszillation zwischen verschiedenen Lösungen zu beheben, passt Adaptive AP die Parameter λ und p im Laufe des iterativen Prozesses dynamisch an. Ein sogenanntes *Moving Window* der Größe w wird verwendet, um zu beobachten, ob innerhalb der letzten w Iterationen Oszillationen aufgetreten sind. Da die Erkennung von Oszillationen sehr komplex ist, beobachtet Adaptive AP, ob eine Lösung Eigenschaften aufweist, die für ein Nicht-Oszillieren sprechen (*Non-Oscillation-Features*). Eine Lösung zeigt dann solche Eigenschaften, wenn die Anzahl der Exemplare im Vergleich zur vorangegangenen Iteration konstant oder gesunken ist. Dabei toleriert das Verfahren Iterationen, welche keine Non-Oscillation-Features aufweisen, zu einem Anteil von einem Drittel innerhalb eines Fensters.

Bei erkannter Oszillation erhöht sich λ um einen Wert von 0,05. Dadurch können Oszillationen ausgelöscht werden. Die Autoren bezeichnen dies als *Adaptive Damping*. Falls λ einen Wert von 0,85 erreicht hat und Oszillationen weiterhin auftreten, verringert sich p um einen konfigurierbaren Wert. Dieser als *Adaptive Escape* bezeichnete Schritt ermöglicht es, einer Oszillation „zu entkommen“, die sich durch Dämpfung nicht auslöschen lässt.

Die Wahl des besten Parameterwertes für p wird in Adaptive AP als *p-scanning* bezeichnet. Wenn das Verfahren konvergiert, validiert es die Lösung mit Hilfe des sogenannten *Silhouette-Index*. Dieser beurteilt das Ergebnis auf Basis der Kompaktheit und Separation der Cluster. Anschließend verkleinert Adaptive AP den Wert von p und setzt das iterative Message-Passing fort, bis es erneut für eine Lösung konvergiert. Dabei wird ein definierbarer Suchraum für p abgearbeitet. Die Schrittweite für die Reduktion von p passt sich dynamisch anhand der Anzahl der Exemplare der aktuellen Lösung an. Adaptive AP gibt jene Lösung aus, die das beste Ergebnis für den Silhouette-Index erzielt hat.

3.6 Hierachical Affinity Propagation

Große Datenmengen sind eine Herausforderung für Clustering-Algorithmen. Bei der Implementierung von AP müssen empfangene Nachrichtenwerte für die Dämpfung in der nächsten Iteration vorgehalten werden. Viele Umsetzungen des Algorithmus, wie die von Frey und Dueck in [13, S. 2] vorgestellte Matlab-Implementieurng oder die Python-Implementierung in der Machine-Learning-Bibliothek *scikit-learn* [33], speichern die Nachrichten in Form von Matrizen. Dazu sind mindestens drei Matrizen,

jeweils eine für die α und die ρ Nachrichten, sowie die Similarity-Matrix erforderlich. Im Fall einer dichtbesiedelten Ähnlichkeitsmatrix (vgl. Abschnitt 3.4.3) und der Verwendung von Gleitkommazahlen mit 4 Byte Speicherbedarf (*Float*) erfordern die Matrizen einen Speicher von $3 \times N \times 4$ Byte für N Datenpunkte [28, S. 895].

AP hat eine Komplexität von $\mathcal{O}(N^2T)$ (vgl. Abschnitt 3.4.3). T steht für die zur Lösungsfindung benötigten Iterationen. Der Zeit- und Speicherbedarf von AP wächst also quadratisch mit der Anzahl der Datenpunkte. Liu et al. stellen *Hierarchical Affinity Propagation* (HAP) [28] vor, um AP auch für umfangreiche Datensammlungen einsetzen zu können. HAP folgt einer Teile-und-herrsche-Strategie (engl. *devide and conquer*). Es gruppiert die Datenpunkte, indem es sie partitioniert und AP auf den verschiedenen Partitionen über verschiedene Hierarchieebenen ausführt.

Abbildung 3.6.1 zeigt das hierarchische Clustering in HAP anhand eines Beispiels für drei Hierarchieebenen h. Mit Hilfe des Parameters M lässt sich die maximale Partitionsgröße einstellen. Auf der ersten Ebene ($h = 1$) zerlegt HAP die Datensammlung zufällig in gleich große Partitionen, deren Größe so nahe wie möglich bei M liegt, aber M nicht überschreitet. Auf jeder dieser Partitionen wird AP ausgeführt. Dabei ergeben sich aus den Cluster-Zentren, welche jede AP-Instanz ermittelt hat, sogenannte *lokale Exemplare*. Die Zuordnung der Datenpunkte zu den lokalen Exemplaren auf den einzelnen Partitionen bleibt ohne Bedeutung. Die lokalen Exemplare werden nun gemischt und ggf. erneut partitioniert, wenn ihre Anzahl die maximale Partitionsgröße überschreitet (vgl. Abb. 3.6.1, $h = 2$). Berechnung von lokalen Exemplaren, Mischen und Partitionierung wiederholen sich nun so lange, bis die Anzahl der lokalen Exemplare den Wert M erreicht oder unterschreitet. Die Ausführung von AP auf der höchsten Hierarchieebene bestimmt letztendlich die sogenannten *globalen Exemplare* der Datensammlung (vgl. Abb. 3.6.1, $h = 3$). Alle

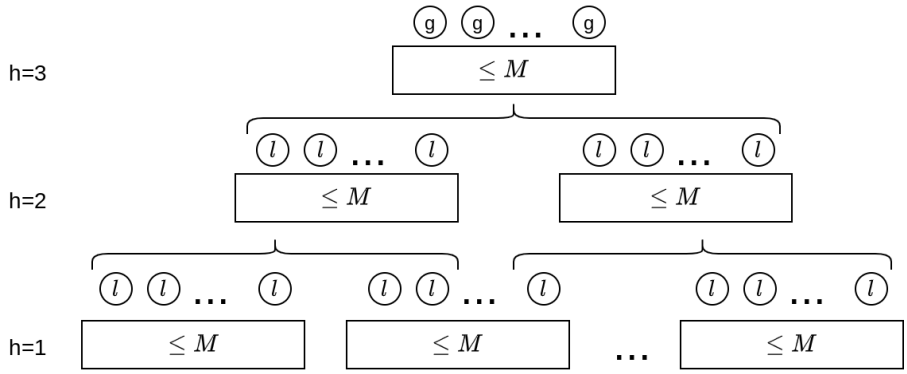


Abbildung 3.6.1: *Hierarchical Affinity Propagation.* Beispielhafte Darstellung für drei Hierarchieebenen h. Lokale Exemplare sind mit einem l gekennzeichnet und globale Exemplare durch ein g. Die Rechtecke stellen Partitionen mit einer Größe $\leq M$ dar.

übrigen Datenpunkte werden den globalen Exemplaren anhand der größten Ähnlichkeit zugeordnet. HAP führt also AP einmal für jede Partition auf jeder Hierarchieebene aus, jeweils mit einer Komplexität von $\mathcal{O}(M^2T)$. T steht dabei für die Anzahl der Iterationen, die AP benötigt, um auf der Partition zu konvergieren.

Im Allgemeinen sind Laufzeit und Qualität des Clustering-Ergebnisses abhängig von der maximalen Partitionsgröße M. Mit einer kleineren Partitionsgröße verringert sich der Zeit- und Speicherbedarf von HAP. Aber je kleiner eine Partition ist, desto weniger repräsentativ ist sie für die Datensammlung und umso weniger Informationen (Ähnlichkeiten zwischen den Datenpunkten) können für das Finden einer guten Clustering-Lösung berücksichtigt werden. Die Wahl von M ist also eine Wahl zwischen Performance und Ergebnisqualtität.

HAP wurde für die vorliegende Masterarbeit auf Basis von Apache Flink implementiert. Liu et al. betrachten in [28] ausschließlich Fälle, bei denen die paarweise Ähnlichkeit eines jeden Datenpunktpaares gegeben ist. Im Fall einer dünnbesetzten Similarity-Matrix kann es vorkommen, dass für einen Datenpunkt zu keinem der globalen Exemplare ein Ähnlichkeitswert bekannt ist. In der Implementierung von HAP für die Evaluation in Kapitel 6 wird davon ausgegangen, dass solche Datenpunkte ein eigenes Cluster bilden. Solche Cluster, die nur aus einem Punkt bestehen, werden als *Singleton* bezeichnet. Je dünner die Similarity-Matrix besetzt und je kleiner der Wert von M ist, desto mehr Datenpunkte müssen Singletons bilden.

Kapitel 4

Konzeptioneller Entwurf

In den beiden vorangegangenen Kapiteln wurde zunächst die Problemstellung der Entwicklung eines Clustering-Verfahrens für MSCD-ER analysiert. Anschließend schuf die ausführliche Erläuterung von ER, Apache Flink, FAMER, dem Max-Sum-Algorithmus und Affinity Propagation die nötigen Grundlagen für den Entwurf und die Implementierung des neuen Verfahrens. Darauf aufbauend widmet sich dieses Kapitel nun dem konzeptionellen Entwurf eines spezialisierten Clustering-Algorithmus für Multi-Source Clean-Dirty Entity Resolution auf Basis von Affinity Propagation.

Zu Beginn beschreibt Abschnitt 4.1 die Anforderungen an den neuen Algorithmus. Anschließend erfolgt in Abschnitt 4.2 der ausführliche Entwurf des MSCD-AP genannten Verfahrens. Eine Modifikation des Faktorgraphen von AP garantiert die Wahrung der Clean-Source-Konsistenz. Diese Modifikation wird ausführlich erläutert, aus dem Max-Sum-Algorithmus hergeleitet und algorithmisch beschrieben. Abschließend beschäftigt sich Abschnitt 4.3 mit zwei verschiedenen Ansätzen, um den neu entworfenen Algorithmus skalierbar für große Datenmengen zu gestalten.

4.1 Anforderungen

Der zu entwerfende Algorithmus soll das in Abschnitt 2.2 vorgestellte MSCD-Clustering umsetzen. Er soll also bei der Gruppierung von Entitätsbeschreibungen berücksichtigen, ob eine Beschreibung aus einer duplikatfreien oder einer duplikatbehafteten Datenquelle stammt. An einen Algorithmus für MSCD-Clustering stellen sich eine Reihe von Anforderungen, die eingehalten werden müssen, um den intuitiven Erwartungen eines Nutzers zu entsprechen und um eine Eignung für das Big-Data-Umfeld zu gewährleisten.

Das Verfahren muss alle Beschreibungen aus der gleichen duplikatfreien Quelle voneinander trennen und unterschiedlichen Clustern zuordnen. Diese Trennung soll strikt und zuverlässig sein. Sie darf nicht aufgrund von Ungewissheiten, scheinbar

besseren Ergebnissen oder durch Heuristiken aufgelockert werden. Das Verfahren muss sämtliche Verteilungen von duplikatfreien und duplikatbehafteten Datenquellen bzw. von deren Entitätsbeschreibungen, berücksichtigen. Dazu gehören auch die Randfälle MS Dirty Clustering und MS Clean Clustering (abgeleitet vom ER-Schema in Abschnitt 2.1), welche als Spezialfälle von MSCD-Clustering anzusehen sind. Das MSCD-Clustering soll MSCD-ER in einem einzigen Durchlauf des ER-Prozesses ermöglichen, damit es einen Vorteil gegenüber der aufeinanderfolgenden Ausführung von MS-Dirty-ER und MS-Clean-ER einbringt (vgl. Abschnitt 2.2). Um einen Einsatz im Big-Data-Umfeld zu ermöglichen, muss der Algorithmus skalierbar sein und performant mit großen Datenmengen umgehen können. Skalierbarkeit bedeutet für den Algorithmen-Entwurf, dass die Komplexität hinsichtlich des Zeit- und Speicherbedarfs möglichst gering gehalten werden soll. Als Ziel sei hier definiert, die quadratische Komplexität $\mathcal{O}(N^2)$ zu unterschreiten.

In der Eingabe des Clusterings müssen alle duplikatfreien Datenquellen eindeutig definiert sein. Die duplikatbehafteten Quellen ergeben sich implizit. Gleiches gilt für die einzelnen Entitätsbeschreibungen. Zumindest für alle Beschreibungen aus duplikatfreien Quellen muss eine Information über die Quellzuordnung gegeben sein. Für Beschreibungen aus duplikatbehafteten Quellen ist dies nicht erforderlich.

4.2 Multi-Source Clean-Dirty Affinity Propagation

Der in Abschnitt 3.4 vorgestellte AP-Algorithmus basiert auf der Anwendung des Max-Sum-Algorithmus auf einem Faktorgraphen. Dieses Konzept dient der sogenannten Constraint-Optimierung, bei der eine Energiefunktion unter Berücksichtigung gewisser Einschränkungen maximiert wird. Das Konzept ist grundsätzlich erweiterbar, indem neue Funktionalitäten durch zusätzliche Constraints hinzugefügt werden. Dies wurde beispielsweise von Amjad et al. in [23] und von Givoni et al. in [17] gezeigt. Diese Erweiterbarkeit soll nun im Folgenden für den Entwurf eines neuen Algorithmus für das MSCD-Clustering genutzt werden. Das neue Verfahren trägt den Namen Multi-Source Clean-Dirty Affinity Propagation (MSCD-AP).

Abschnitt 4.2.1 beschreibt ein neues Constraint namens *Clean-Source-Constraint*. Es stellt sicher, dass Datenpunkte aus der gleichen „sauberen“ Quelle korrekt voneinander getrennt werden. Das Clean-Source-Constraint erweitert den Faktorgraph von AP um zusätzliche Faktorknoten. Abschnitt 4.2.2 leitet die neuen Nachrichtenformeln für das Message-Passing auf dem Faktorgraphen aus dem Max-Sum-Algorithmus her. In Abschnitt 4.2.3 folgt schließlich eine ausführliche Beschreibung der algorithmischen Umsetzung des zuvor hergeleiteten Nachrichtenaustauschs.

4.2.1 Das Clean-Source-Constraint

MSCD-AP muss sicherstellen, dass es Datenpunkte aus der gleichen „sauberen“ Quelle nicht dem gleichen Cluster zuordnet. Um dies zu garantiert, wird AP um eine zusätzliche Einschränkung mit dem Namen *Clean-Source-Constraint* erweitert. Die Erweiterung nutzt ebenfalls das in Abschnitt 3.4.3 vorgestellte Binärvariablenmodell. Die Clusterzuordnung lässt sich in diesem Modell an der Binärmatrix $\mathbb{B}$ ablesen. Die Exemplarzuordnung jedes Datenpunktes wird durch jeweils eine Zeile repräsentiert. Alle Punkte des gleichen Clusters sind dem gleichen Exemplar zugeordnet. Sie haben also in der gleichen Spalte von $\mathbb{B}$ einen Wert von eins. Das Clean-Source-Constraint muss nun für jede duplikatfreie Quelle und jede Spalte in $\mathbb{B}$ prüfen, dass eine Spalte nicht für mehr als einen Punkt der Quelle eine Eins enthält.

Zur Veranschaulichung zeigt Tabelle 4.2.1 die Cluster-Zuordnung des Einführungsbeispiels aus Abbildung 2.2.2a. Die beiden Datenpunkte eins und vier stammen aus der gleichen Quelle X und sind dem gleichen Cluster zugeordnet, welches Punkt zwei als Exemplar repräsentiert. Falls Quelle X duplikatfrei ist, würde dieses Clustering das Clean-Source-Constraint verletzen. Punkt eins oder Punkt vier müssten einem anderen Exemplar zugeordnet werden. Zur Gewährleistung des Clean-Source-Constraints muss der Algorithmus nur die grün hervorgehobenen Zellen für jede Spalte überprüfen. Wenn Quelle Y Duplikate enthält, sind die weiß hinterlegten Zellen für die Einschränkung irrelevant. Bei Duplikatfreiheit von Quelle Y müsste die Variablenbelegung in den Zeilen null und drei zusätzlich überwacht werden. Die Variablen müssen sich nur innerhalb ihrer Quelle abgleichen und nicht mit den Zeilen der anderen duplikatfreien Quellen.

Für das Clean-Source-Constraint lässt sich eine Funktion entsprechend Gleichung 4.2.1 formulieren, welche die Faktorisierung der Energiefunktion erweitert. Sie prüft

	0Y	**1X**	**2Z**	**3Y**	**4X**
0Y	**1**	0	0	0	0
1X	0	0	**1**	0	0
2Z	0	0	**1**	0	0
3Y	0	0	0	**1**	0
4X	0	0	**1**	0	0

Tabelle 4.2.1: *Binärmatrix zur Veranschaulichung des Clean-Source-Constraints.* Das dargestellte Clustering entspricht dem Einführungsbeispiel aus Abbildung 2.2.2a. Die Quellen der Datenpunkte sind durch die Buchstaben X bis Z in der Kopfzeile gekennzeichnet.

die Summe aller Zellen der Spalte j und der „sauberen“ Quelle Q. Ist die Summe größer als eins, liegt eine Verletzung des Clean-Source-Constraints vor. Die Funktion gibt dann einen Wert von $-\infty$ aus, der auf die Energiefunktion addiert wird. Dadurch kann die Energie ihr Optimum nicht unter einer fehlerhaften Variablenbelegung erreichen. Ist die Summe kleiner oder gleich dem Wert eins, gibt die Funktion einen Wert von null aus und lässt damit die Energie unverändert.

Die Energiefunktion für MSCD-AP ergibt sich entsprechend Formel 4.2.2. Die Faktorisierung wird für jede duplikatfreie Quelle Q und jede Spalte j um die Funktion t_{Qj} erweitert. An dieser Stelle sei nochmals darauf hingewiesen, dass weiterhin von einer Faktorisierung die Rede ist, obwohl es sich bei der Energiefunktion um eine Summe handelt. Das Konzept Faktorgraph basiert ursprünglich auf einem Produkt der durch die Faktorknoten repräsentierten Funktionen. Die Verwendung einer Summe anstelle eines Produkts beim Max-Sum-Algorithmus ändert die Bezeichnung des Konzeptes nicht.

$$t_{Qj}(\mathbb{B}(i \in Q, j) = \begin{cases} 0 & \text{wenn } \sum_{i \in Q} b_{ij} \leq 1 \\ -\infty & \text{andernfalls} \end{cases} \tag{4.2.1}$$

$$E(\mathbb{B}) = \sum_{ij} s_{ij} b_{ij} + \sum_{i} g_i(\mathbb{B}(i,:)) + \sum_{j} h_j(\mathbb{B}(:,j)) + \sum_{Q} \sum_{i \in Q,\, j} t_{Qj}(\mathbb{B}(i,j)) \tag{4.2.2}$$

Entsprechend der zusätzlichen Faktorfunktionen erfolgt für MSCD-AP eine Erweiterung des Faktorgraphen von AP. Abbildung 4.2.1 zeigt dies anhand des Einführungsbeispiels, welches zuvor für Tabelle 4.2.1 erläutert wurde. In diesem Fall wird davon ausgegangen, dass die beiden Quellen X und Y duplikatfrei vorliegen. Jede Spalte j der im Graph abgebildeten Binärmatrix erhält einen zusätzlichen Faktorknoten t_{Qj} für jede saubere Quelle Q. Diese sind mit jenen Variablenknoten b_{ij} der Spalte j durch eine Kante verbunden, deren Zeile i für einen Datenpunkt aus der Quelle Q des Faktorknotens steht. Im Beispiel erhält der Graph also in jeder Spalte j jeweils einen zusätzlichen Faktorknoten t_{Xj} bzw. t_{Yj}, um das Clean-Source-Constraint für die sauberen Datenquellen X und Y zu wahren.

Das neue Clean-Source-Constraint könnte mit dem Exemplar-Consistency-Constraint in Konflikt geraten. Letzteres zwingt einen Datenpunkt, sich selbst als Exemplar zu wählen, wenn er von einem anderen Punkt gewählt wurde. Die diagonale Variable b_{jj} der Spalte j muss also einen Wert von eins haben, falls eine andere Variable der Spalte eine Eins enthält. Das Clean-Source-Constraint hingegen würde für b_{jj} einen Wert von null erzwingen, falls der andere Punkt aus der gleichen „sauberen“

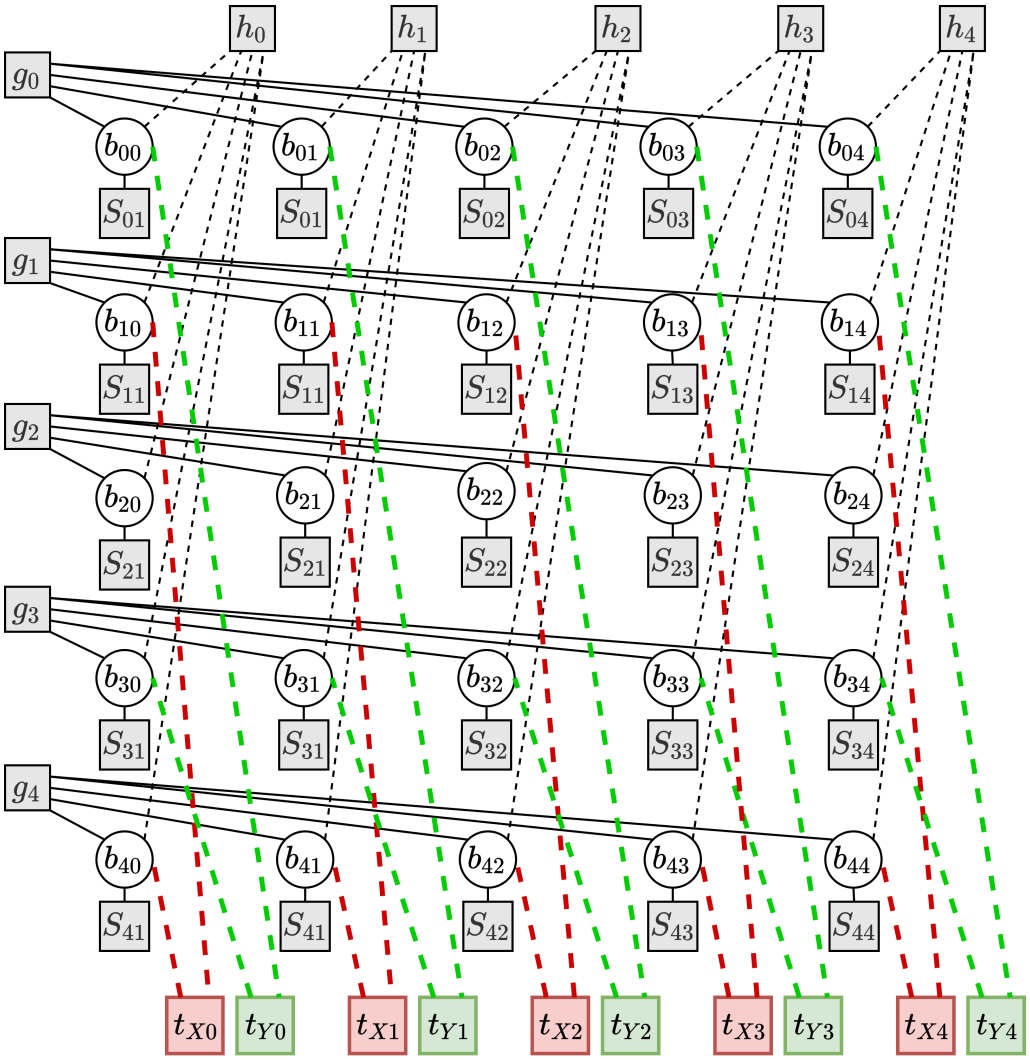


Abbildung 4.2.1: *Faktorgraph für MSCD-AP.* Dargestellt ist der Graph für das Einführungsbeispiel in Abbildung 2.2.2b. Die Quellen X und Y sind duplikatfrei.

Quelle wie das Exemplar stammt. Beide Constraints verlangen also einen anderen Wert für b_{jj}. Dies gefährdet die Konsistenz der Binärmatrix bezüglich der Constraints. Um solche Situationen zu vermeiden, müssen alle Ähnlichkeiten zwischen Datenpunkten der gleichen duplikatfreien Quelle aus dem Similarity-Graph entfernt werden. Dies kann bereits beim Entity-Matching im ER-Prozess oder in einem Vorverarbeitungsschritt von MSCD-AP erfolgen. Auf diese Weise ist sichergestellt, dass ein Punkt keinen anderen als Exemplar wählen kann, der aus der gleichen duplikatfreien Quelle stammt. Damit kann der Konfliktfall zwischen Clean-Source-Constraint und Exemplar-Consistency-Constraint nicht eintreten.

4.2.2 Message-Passing in MSCD-AP

Zur Maximierung der Energiefunktion führt MSCD-AP einen Nachrichtenaustausch auf dem Faktorgraphen aus. Die Formeln zur Berechnung der Nachrichtenwerte ergeben sich aus dem in Abschnitt 3.4.2 vorgestellten Max-Sum-Algorithmus. Abbildung 4.2.2 zeigt die in MSCD-AP zwischen Faktor- und Variablenknoten ausgetauschten Nachrichten. Die schwarz dargestellten Nachrichtenformeln für η_{ij} und α_{ij} bleiben im Vergleich zum Basisalgorithmus AP unverändert (vgl. Formel 3.4.12 und 3.4.13). Ebenso enthält die Nachricht s_{ij} weiterhin die Ähnlichkeit von Datenpunkt i zu Punkt j. Die übrigen Nachrichtenformeln sind für MSCD-AP im Vergleich zum Basisalgorithmus modifiziert (violett) bzw. gänzlich neu (orange).

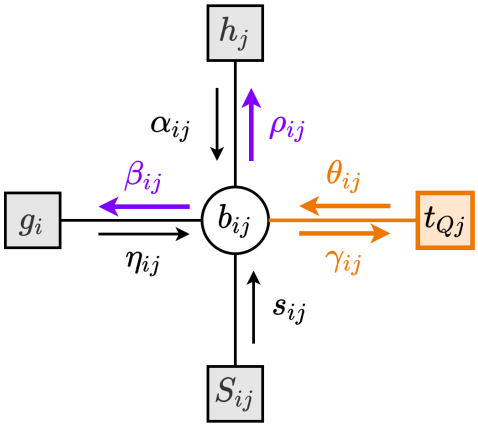


Abbildung 4.2.2: Nachrichten auf einem Faktorgraph für MSCD-AP

$$\beta_{ij} = s_{ij} + \alpha_{ij} + \theta_{ij} \tag{4.2.3}$$

$$\rho_{ij} = s_{ij} + \eta_{ij} + \theta_{ij} \tag{4.2.4}$$

$$\gamma_{ij} = s_{ij} + \alpha_{ij} + \eta_{ij} \tag{4.2.5}$$

$$\theta_{ij} = min(0, -\max_{k \neq i}[\gamma_{kj}]) \tag{4.2.6}$$

β_{ij}, ρ_{ij} und γ_{ij} sind Nachrichten von einem Variablen- zu einem Faktorknoten. Sie lassen sich aus Formel 3.4.2 des Max-Sum-Algorithmus herleiten und sind definiert als die Summe der eingehenden Nachrichten aller Nachbarn des Variablenknotens, ausgenommen des Faktorknotens, zu dem die Nachricht gesendet werden soll. Somit ergeben sich β_{ij}, ρ_{ij} und γ_{ij} entsprechend der Formeln 4.2.3 bis 4.2.5. θ_{ij} ist als Nachricht von einem Faktor- zu einem Variablenknoten deutlich komplexer herzuleiten. Im restlichen Abschnitt erfolgt eine ausführliche Entwicklung von Formel 4.2.6 anhand von Formel 3.4.3 des Max-Sum-Algorithmus. Die Ableitung richtet sich dabei nach Givonis Herleitung der Nachrichtenformeln von AP für das Binärvariablenmodell [16, S. 54-59].

Im Binärvariablenmodell gibt es zwei mögliche Fälle, die zu berücksichtigen sind. Der Variablen des Knotens b_{ij}, zu dem vom Faktorknoten t_{Qj} eine Nachricht gesendet wird, kann ein Wert von eins oder null zugewiesen sein. Zunächst werden die beiden Fälle separat voneinander hergeleitet und anschließend zu einer gemeinsamen

Formel kombiniert. Für beide Fälle müssen *alle möglichen Kombinationen* für die Variablenbelegung *aller anderen* benachbarten Knoten $b_{kj}, k \neq i$ von t_{Qj} berücksichtigt werden. Jede Konfiguration der Variablen, die das Clean-Source-Constraint verletzt, ergibt bei der Evaluation durch die t_{Qj} Funktion ein Ergebnis von $-\infty$. Solche Konfigurationen sind im Sinne der Energiefunktion von MSCD-AP nicht optimal und müssen verhindert werden. Jede ausgehende Nachricht eines Constraint-Faktorknotens berücksichtigt also implizit nur die Variablenbelegungen, welche das Constraint einhalten.

Herleitung für $b_{ij} = 1$:

Wenn für den Variablenknoten b_{ij} ein Wert von eins gesetzt ist, dann ist Datenpunkt j das Exemplar für Punkt i. Entsprechend der Definition des Faktorgraphen von MSCD-AP im vorangegangenen Abschnitt tauscht t_{Qj} nur Nachrichten mit Variablenknoten aus, welche die Exemplarzuordnung von Datenpunkten der gleichen Quelle Q repräsentieren. Dabei sei q die Anzahl der Elemente aus der duplikatfreien Quelle Q. Aus Formel 3.4.3 des Max-Sum-Algorithmus ergibt sich für θ_{ij} in dem Fall von $b_{ij} = 1$ folgende Nachrichtenformel:

$$\theta_{ij}(1) = \max_{b_{kj}, k \neq i} [ln\, t_{Qj}(b_{1j}, ..., b_{ij} = 1, ..., b_{qj}) + \sum_{b_{kj}, k \neq i} \mu_{b_{kj} \to t_j}(b_{kj})]. \tag{4.2.7}$$

Alle in t_{Qj} eingehenden Nachrichten $\mu_{b_{kj} \to t_{Qj}}(b_{kj})$ sind in Abbildung 4.2.2 als $\gamma_{kj}(b_{kj})$ definiert. Aus Formel 4.2.7 ergibt sich damit Gleichung 4.2.8.

$$\theta_{ij}(1) = \max_{b_{kj}, k \neq i} [ln\, t_{Qj}(b_{1j}, ..., b_{ij} = 1, ..., b_{qj}) + \sum_{b_{kj}, k \neq i} \gamma_{kj}(b_{kj})] \tag{4.2.8}$$

Um das Clean-Source-Constraint nicht zu verletzen, darf kein anderer Punkt in Q ebenfalls j als Exemplar wählen. Dadurch wären zwei Objekte der gleichen sauberen Quelle im selben Cluster vertreten. Deshalb werden in Formel 4.2.9 alle anderen benachbarten Variablenknoten $b_{kj}, k \neq i$ von t_{Qj} auf einen Wert von null gesetzt.

$$\theta_{ij}(1) = \max_{b_{kj}, k \neq i} [ln\, t_{Qj}(b_{1j} = 0, ..., b_{ij} = 1, ..., b_{qj} = 0) + \sum_{b_{kj}, k \neq i} \gamma_{kj}(b_{kj} = 0)] \tag{4.2.9}$$

Diese Variablenkonfiguration ist die einzige, die das Clean-Source-Constraint einhält, wenn b_{ij} auf eins gesetzt ist. Folglich ist sie die optimale Konfiguration, welche bei der Evaluation durch die t_{Qj} Funktion *einen Wert von null* ergibt (vgl. Formel 4.2.10).

Damit erlaubt die Funktion von t_{Qj}, dass die Energiefunktion von MSCD-AP für diese Variablenkonfiguration ihr Maximum erreichen kann.

$$t_{Qj}(b_{1j} = 0, ..., b_{ij} = 1, ..., b_{qj} = 0) = 0 \tag{4.2.10}$$

Da es nur eine mögliche Variablenkombination gibt, entfällt die Maximierung aus Formel 4.2.9. Die Nachricht für den Fall $b_{ij} = 1$ ist also:

$$\theta_{ij}(1) = \sum_{k \neq i} \gamma_{kj}(0). \tag{4.2.11}$$

Herleitung für $b_{ij} = 0$:

Aus Formel 3.4.3 des Max-Sum-Algorithmus ergibt sich für θ_{ij} in dem Fall $b_{ij} = 0$ die Nachrichtenformel:

$$\theta_{ij}(0) = \max_{b_{kj}, k \neq i} [ln\, t_{Qj}(b_{1j}, ..., b_{ij} = 0, ..., b_{qj}) + \sum_{b_{kj}, k \neq i} \gamma_{kj}(b_{kj})]. \tag{4.2.12}$$

Wenn Datenpunkt j nicht das Exemplar für Punkt i darstellt, gibt es mehr Möglichkeiten für die Konfiguration der anderen benachbarten Variablen von t_{Qj}. Maximal eine der Variablen $b_{kj}, k \neq i$ darf auf einen Wert von eins gesetzt sein, um das Clean-Source-Constraint einzuhalten. Es ist also auch möglich, dass alle $b_{kj}, k \neq i$ mit einem Wert von null belegt sind. Dies ist der Fall, wenn keiner der anderen Datenpunkte aus Quelle Q dem Exemplar j zugeordnet ist oder, wenn j selbst kein Exemplar ist. Es existieren q mögliche Lösungen, die das Clean-Source-Constraint einhalten: $q - 1$ für jede Variable in $b_{kj}, k \neq i$ (-1, weil i auch in Q enthalten ist), die jeweils als einzige mit dem Wert eins belegt ist und eine Lösung für den Fall, dass alle Variablen $b_{kj}, k \neq i$ den Wert null enthalten.

Zunächst wird der Fall betrachtet, dass alle $b_{kj}, k \neq i$ mit dem Wert null belegt sind. Dieser Fall sei x. Da das Clean-Source-Constraint eingehalten wird, evaluiert die Funktion von t_{Qj} die Variablenbelegung mit dem Ergebnis null. Es ergibt sich die Formel 4.2.13, welche der zuvor hergeleiteten Formel 4.2.11 entspricht.

$$x = 0 + \sum_{k \neq i} \gamma_{kj}(0) \tag{4.2.13}$$

Der Fall, in dem exakt eine der Variablen in $b_{kj}, k \neq i$ mit dem Wert eins belegt ist, sei mit y bezeichnet und in Formel 4.2.14 beschrieben.

$$y = \max_{k \neq i}[0 + \gamma_{kj}(1) + \sum_{p \notin \{k,i\}} \gamma_{pj}(0)] \tag{4.2.14}$$

Aus Formel 4.2.12 ergibt sich die Nachricht für $b_{ij} = 0$ als Maximum der beiden Fälle x und y in der Gleichung 4.2.15.

$$\theta_{ij}(0) = \max(x, y) \tag{4.2.15}$$

Kombination von $\theta_{ij}(1)$ und $\theta_{ij}(0)$

Da jede Variable b_{ij} binär ist, erscheint es, als müssten zweiwertige Nachrichten zwischen den Faktor- und Variablenknoten in AP ausgetauscht werden. Laut Givoni genügt es jedoch für AP, die Differenz der Nachrichtenwerte für die beiden möglichen Belegungen von b_{ij} zu berechnen und als Nachricht auszutauschen [16, S. 53]. Die ursprüngliche zweiwertige Nachricht ließe sich aus dem Skalar bis zu einer additiven Konstante rekonstruieren. Diese sei aber unwichtig, da die Addition einer Konstante auf alle Nachrichtenwerte das Ergebnis des Algorithmus nicht verändert. Givoni folgend ergibt sich daher die von t_{Qj} gesendete Nachricht in Formel 4.2.16 aus der Differenz der beiden Nachrichtenformeln für die beiden möglichen Belegungen der Binärvariablen b_{ij}.

$$\theta_{ij} = \theta_{ij}(1) - \theta_{ij}(0) \tag{4.2.16}$$

Die Nachricht für den Fall $b_{ij} = 1$ ist identisch mit x (vgl. Formel 4.2.11 und Formel 4.2.13). Durch Ersetzung von $\theta_{ij}(1)$ mit x und $\theta_{ij}(0)$ entsprechend Formel 4.2.15 ergibt sich

$$\theta_{ij} = x - \max(x, y) \tag{4.2.17}$$

und mit Hilfe der Transformation

$$x - \max(x, y) = min(0, x - y) \tag{4.2.18}$$

entfällt das zweite x. Durch Einsetzen von x aus Formel 4.2.13 und y aus Formel 4.2.14 entsteht die folgende Gleichung:

$$\theta_{ij} = min(0, \sum_{k \neq i} \gamma_{kj}(0) - \max_{k \neq i}[\gamma_{kj}(1) + \sum_{p \notin \{k,i\}} \gamma_{pj}(0)]). \tag{4.2.19}$$

Die allgemeine Transformation

$$a - \max_{b \in C}(b) = - \max_{b \in C}(b - a) \tag{4.2.20}$$

bringt die Komponente $\sum_{k \neq i} \gamma_{pj}(0)$ in die Maximierung, so dass Formel 4.2.21 entsteht.

$$\theta_{ij} = min(0, - \max_{k \neq i}[\gamma_{kj}(1) + \sum_{p \notin \{k,i\}} \gamma_{pj}(0) - \sum_{k \neq i} \gamma_{kj}(0)]) \tag{4.2.21}$$

Bei der Subtraktion beider Summen in Formel 4.2.21 bleibt in Gleichung 4.2.22 nur $-\gamma_{kj}(0)$ übrig, da es nicht in der ersten Summe enthalten ist.

$$\theta_{ij} = min(0, - \max_{k \neq i}[\gamma_{kj}(1) - \gamma_{kj}(0)]) \tag{4.2.22}$$

So wie θ_{ij} in Formel 4.2.16 ist auch γ_{kj} die Differenz der beiden Nachrichtenwerte $\gamma_{kj}(1)$ und $\gamma_{kj}(0)$ für die beiden möglichen Belegungen der Binärvariable. Damit ergibt sich abschließend Formel 4.2.23 für die Nachricht des Clean-Source-Constraints von Faktorknoten t_{Qj} zum Variablenknoten b_{ij} in MSCD-AP.

$$\theta_{ij} = min(0, - \max_{k \neq i}[\gamma_{kj}]). \tag{4.2.23}$$

4.2.3 Algorithmische Betrachtung von MSCD-AP

In den vorhergehenden Abschnitten wurde MSCD-AP zunächst als eine Erweiterung von AP um das Clean-Source-Constraint entworfen. Anschließend erfolgte eine Herleitung der Nachrichtenformeln für das Max-Sum-Message-Passing, mit dessen Hilfe MSCD-AP das Optimierungsproblem des optimalen Clusterings approximiert. Dieser Abschnitt widmet sich nun der algorithmischen Betrachtung von MSCD-AP. Er zeigt, wie das Verfahren aus den Datenpunktähnlichkeiten und den weiteren Parametern der Eingabe schließlich eine Clustering-Lösung ermittelt. Der Pseudocode in Algorithmus 4.2.1 dient der Veranschaulichung des Verfahrens. Er zeigt eine mögliche Umsetzung von MSCD-AP mit Hilfe von Matrizen und ist grob angelehnt an Frey und Duecks Matlab-Implementierung von AP in [13, S. 2].

Die Nachrichten des Message-Passing auf dem Faktorgraph aus Abbildung 4.2.2 werden auf einzelne $n \times n$ Matrizen abgebildet. n steht für die Anzahl der zu gruppierenden Datenpunkte. Jeder Nachrichtentyp erhält eine eigene Matrix. Matrizen sind als groß geschriebene Variablen zu erkennen. A enthält die α Nachrichten, R jene von ρ, G die von γ und T die Nachrichten für θ. *Beta* ist selbsterklärend. $\mathbb{B}$ repräsentiert die Binärmatrix und S die Ähnlichkeitsmatrix. Die Nachricht α_{ij} wird auf

Zeile i und Spalte j der Matrix A abgebildet. Für die anderen Nachrichten verhält es sich analog. Die Notation $Beta[i,:]$ steht für eine Aktion, die über alle Spalten der Zeile i ausgeführt wird. $G[src(:),j]$ beschreibt eine Aktion in Spalte j, ausgeführt über alle Zeilen, welche Datenpunkte der Quelle *src* repräsentieren. Mit dieser einfachen Abbildung der Nachrichtenwerte auf Matrizen lässt sich MSCD-AP sowohl für eine dicht- als auch für eine dünnbesetzte Similarity-Matrix umsetzen (vgl. Abschnitt 3.4.3). Bei einer dünnbesetzten Ähnlichkeitsmatrix berechnet MSCD-AP in diesem Ansatz zwar auch Nachrichtenwerte für Kanten des Faktorgraphen, die eigentlich aufgrund der fehlenden Ähnlichkeiten eingespart werden könnten. Dies hat jedoch keinen Einfluss auf das Ergebnis des Clusterings. Für die fehlenden Ähnlichkeiten wird ein Wert von null angenommen. Ein Datenpunkt j, zu dem Punkt i eine Ähnlichkeit von null aufweist, kommt nicht als Exemplar für i infrage. Es ergibt sich bei der Maximierung der Energiefunktion also keine Variablenkombination mit $b_{ij} = 1$. In Abschnitt 4.3.2 wird später ein Entwurf vorgestellt, der sich eine dünnbesetzte Similarity-Matrix zum Vorteil macht und die nicht benötigten Nachrichten einspart.

In MSCD-AP ist der Preference-Parameter getrennt für Datenpunkte aus duplikatfreien (p_{CS}) und duplikatbehafteten (p_{DS}) Quellen einstellbar. Bei den Experimenten für die Evaluation von MSCD-AP in Kapitel 6 zeigte sich, dass eine Konfiguration mit $p_{CS} > p_{DS}$ die Arbeit des Algorithmus erleichtern kann. Dies entspricht einer Bevorzugung von Datenpunkten aus „sauberen“ Quellen für die Auswahl der Exemplare. MSCD-AP konvergiert in weniger Iterationen und kann ein besseres Clustering-Ergebnis erzielen. Datenpunkte einer „sauberen“ Quelle müssen voneinander getrennt werden und eignen sich daher gut als Cluster-Zentrum.

Die Similarity-Matrix S, der Dämpfungsfaktor *dmp*, die Preference-Werte p_{CS} und p_{DS}, der Grad des Rauschens (*noiseLevel*) und die Information *srcInfo*, welcher Datenpunkt aus welcher Quelle stammt, bilden die Eingabe des Algorithmus. Dabei ist die Quellinformation nur für Punkte aus „sauberen“ Quellen nötig (vgl. Anforderungen in Abschnitt 4.1). *SrcInfo* ist als ein Mapping von Datenquellnamen anzusehen, die auf Arrays mit Datenpunkt-IDs abgebildet werden. Dabei enthält ein Array die IDs aller Datenpunkte jener Quelle, der es zugeordnet ist. Das Ergebnis von MSCD-AP ist die Binärmatrix $\mathbb{B}$, welche die Exemplarzuordnung jedes Datenpunktes enthält. Die Matrizen aller Nachrichten, ausgenommen der eingegebenen Ähnlichkeitsmatrix, werden in Zeile 1 des Algorithmus initialisiert. Jede Zelle enthält zu Beginn einen Wert von null. Die diagonalen Elemente von S enthalten die Selbstähnlichkeit eines Datenpunktes. Zeile 4 setzt sie entsprechend der Preference-Parameter. Nach dem Verrauschen der Ähnlichkeitsmatrix in Zeile 5 beginnt das

iterative Message-Passing. Rauschen und Dämpfung dienen der Prävention von Oszillationen (vgl. Abschnitt 3.4.3).

Innerhalb einer Iteration werden jeweils zwei Nachrichten für jede Kante des Faktorgraphen berechnet, eine für jede Richtung. Den ersten Schritt bildet die Berechnung der β-Nachrichten in Zeile 7. Sie ergeben sich aus der Addition der Matrizen A, S und T. Da A und T in der ersten Iteration in jeder Zelle eine null enthalten, entspricht $Beta$ zu Beginn S. Die Zeilen 8 bis 11 zeigen die Berechnung der η-Nachrichten. In E ergibt sich jede Zelle einer Zeile aus der Negierung des größten Wertes *aller anderen* Zellen der gleichen Zeile in $Beta$. Um dies zu erreichen, werden der größte und der zweitgrößte Wert der $Beta$-Zeile bestimmt. Die Zelle von E, in der die $Beta$-Zeile maximal ist, erhält die Negation des zweitgrößten Wertes. Alle anderen werden auf die Negierung des größten Wertes gesetzt.

Zeile 12 berechnet G als Summe von E, A und S. In den Zeilen 13 bis 17 erfolgt die Kalkulation der θ-Nachrichten. MSCD-AP behandelt jede „saubere“ Quelle

Algorithmus 4.2.1 : Pseudocode für MSCD-AP

Eingabe : S, dmp, p_{DS}, p_{CS}, $noiseLevel$, $srcInfo$
Ergebnis : $\mathbb{B}$ mit Exemplarzuordnungen

1 $n \leftarrow size(S)$; $A \leftarrow zeros(n,n)$; $R \leftarrow zeros(n,n)$; $T \leftarrow zeros(n,n)$; // init msgs
2 $n \leftarrow n-1$;
3 **for** $i = 0 : n$ **do** // set preferences..
4 **if** $i \in srcInfo$ **then** $S[i][i] \leftarrow p_{CS}$ **else** $S[i][i] \leftarrow p_{DS}$; // ..by src type
5 $S \leftarrow S + gaussianRandom() \cdot noiseLevel$; // add noise
6 **for** $iter = 0 : iterMax$ **do**
7 $Beta \leftarrow A + S + T$;
8 **for** $i = 0 : n$ **do** // compute Eta
9 $rowMax, rowMaxId, rowScndMax \leftarrow getRowMaxValues(Beta[i,:])$;
10 $E[i,:] \leftarrow -1 \cdot rowMax$;
11 $E[i, rowMaxId] \leftarrow -1 \cdot rowScndMax$;
12 $G = E + A + S$;
13 **for** $src \in srcInfo$ **do** // compute Theta
14 **for** $j = 0 : n$ **do**
15 $colMax, colMaxId, colScndMax \leftarrow getColMaxValues(G[src(:), j])$;
16 $T[src(:), j] \leftarrow min(0, -1 \cdot colMax)$;
17 $T[colMaxId, j] \leftarrow min(0, -1 \cdot colScndMax)$;
18 $R_{new} = E + T + S$;
19 $R = dmp \cdot R + (1 - dmp) \cdot R_{new}$; // damping of Rho
20 **for** $j = 0 : n$ **do** // compute Alpha
21 $colSum \leftarrow sum(max(0, R[:, j])) - max(0, R[j, j])$;
22 **for** $i = 0 : n$ **do** $A_{new}[i, j] \leftarrow min(0, R[j, j] + colSum - max(0, R[i, j])$;
23 $A_{new}[j, j] \leftarrow colSum$;
24 $A \leftarrow dmp \cdot A + (1 - dmp) \cdot A_{new}$; // damping of Alpha
25 $\mathbb{B} \leftarrow (A + R) > 0$; // set binary matrix
26 **if** $checkConvergence(\mathbb{B})$ **then** break;

(*src*) einzeln und legt einen Filter auf die Zeilen von G und T, so dass im Folgenden nur die Zeilen der beiden Matrizen berücksichtigt werden, welche aus der jeweiligen Quelle *src* stammen. Für jede Spalte in T ergibt sich der Wert einer Zelle aus der Negation des größten Wertes *aller anderen* Zellen der gefilterten G-Matrix in der gleichen Spalte. Dazu wird wieder, analog zur η-Kalkulation, der größte und zweitgrößte Wert der Spalte bestimmt. Anders als bei E werden für T jedoch nur negative Werte gesetzt. Die T-Matrix enthält anschließend nur in jenen Zeilen Werte ungleich null, welche Datenpunkte aus duplikatfreien Quellen repräsentieren. Wenn MSCD-AP keine „sauberen“ Quellen im Parameter *srcInfo* übergeben werden, entfällt die Berechnung von T. Das Ergebnis des Algorithmus entspricht dann dem des klassischen AP ohne Clean-Source-Constraint.

In den Zeilen 18 und 19 errechnet sich R aus der Summe von E, T und S, gedämpft um einen Anteil von *dmp* des Ergebnisses der letzten Iteration. Für die α Kalkulation müssen, entsprechend Formel 3.4.13, die diagonalen Zellen $A[j, j]$ anders als die restlichen Elemente behandelt werden. Der Algorithmus ermittelt zunächst die Spaltensummen für jede Spalte in R (Zeile 21). Diese berücksichtigen jedoch nur positive Zellen und exkludieren das Diagonalelement $R[j, j]$. Zeile 23 setzt die diagonalen Zellen $A_{new}[j, j]$ auf die jeweilige Spaltensumme. Die restlichen Zellen $A_{new}[i, j]$ ergeben sich ebenfalls aus der Spaltensumme (Zeile 22). Dabei wird jedoch der eigene Wert aus $R[i, j]$ exkludiert und der Diagonalwert ohne die Filterung auf positive Zellen hinzugefügt. Falls das Resultat größer als null ist, erhält $A_{new}[i, j]$ einen Wert von null. So wie für R dämpft Zeile 24 die Matrix A entsprechend des Dämpfungsfaktors um das Ergebnis der letzten Iteration.

Am Ende jeder Iteration bestimmt MSCD-AP die Binärmatrix $\mathbb{B}$. Sie enthält eine Eins in allen Zellen $\mathbb{B}[i, j]$, bei denen die Summe aus $A[i, j]$ und $R[i, j]$ positiv ist, und eine Null in den übrigen Zellen. Die Konvergenzprüfung ist in Zeile 26 vereinfacht durch die Funktion *checkConvergence()* dargestellt. Wenn die diagonalen Elemente von $\mathbb{B}$ über eine gewisse Zahl von Iterationen unverändert bleiben, dann ist MSCD-AP konvergiert. Der Algorithmus beendet sich und gibt mit $\mathbb{B}$ das ermittelte Clustering aus.

Das in Algorithmus 4.2.1 vorgestellte MSCD-AP hat ebenso wie der Basisalgorithmus AP eine Komplexität von $\mathcal{O}(N^2T)$. N steht für die Anzahl der Datenpunkte und T für die zur Lösungsfindung benötigten Iterationen. Die zusätzlichen Berechnungen für das Clean-Source-Constraint ändern die obere Schranke der Komplexitätsklasse des Algorithmus nicht.

4.3 Ansätze zur Steigerung der Skalierbarkeit von MSCD-AP

Entsprechend der Anforderungen in Abschnitt 4.1 soll MSCD-AP für das Clustering großer Datenmengen im Big-Data-Umfeld einsetzbar sein. In Abschnitt 3.6 wurde die schlechte Skalierbarkeit des Basisalgorithmus AP diskutiert, dessen Zeit- und Speicherbedarf quadratisch mit der Anzahl der zu gruppierenden Datenpunkte wächst. Gleichermaßen verhält es sich auch für das abgeleitete MSCD-AP. Der folgende Abschnitt befasst sich daher mit der Erarbeitung von Lösungsansätzen, um den Algorithmus skalierbar zu gestalten. Der erste Ansatz, Hierarchical MSCD-AP (MSCD-HAP), ermöglicht Skalierbarkeit für dichtbesetzte (vgl. Abschnitt 3.4.3) Ähnlichkeitsmatrizen, jedoch unter Beeinträchtigung der Clustering-Qualtität aufgrund von Informationsverlust. Der zweite Ansatz, Sparse MSCD-AP, erlaubt eingeschränkte Skalierbarkeit für den Fall einer dünnbesetzten Similarity-Matrix.

4.3.1 Hierarchical MSCD-AP (MSCD-HAP)

Das in Abschnitt 3.6 vorgestellte Teile-und-Herrsche-Verfahren HAP lässt sich nicht ohne Weiteres für MSCD-AP anwenden. HAP ermittelt globale Exemplare für große Datensammlungen durch Partitionierung und die Ausführung von AP in einer hierarchischen Struktur. Anstelle von AP könnte einfach MSCD-AP auf den einzelnen Partitionen ausgeführt werden. Doch die Exemplarzuordnung von MSCD-AP, welche Datenpunkte aus der gleichen duplikatfreien Quelle voneinander trennt, ginge in der hierarchischen Partitionierung verloren. In jeder Hierarchieebene sind nur die lokalen Exemplare relevant und die Cluster-Zuordnung wird verworfen. Im letzten Schritt von HAP werden alle anderen Datenpunkte den globalen Exemplaren anhand der größten Ähnlichkeit zugeordnet. Doch dies garantiert nicht die Trennung von Datenpunkten aus der gleichen duplikatfreien Quelle.

Ein einfaches Bewahren der Cluster-Zuordnung über die Hierarchie würde das Problem nicht lösen. Das Clustering von lokalen Exemplaren auf einer höheren Hierarchieebene könnte die Clean-Source-Konsistenz verletzen, da zwei lokale Exemplare zusammen gruppiert sein könnten, obwohl ihnen auf der vorherigen Ebene Datenpunkte aus der gleichen „sauberen“ Quelle zugeordnet wurden. Um dies zu verhindern, könnten Datenpunkte ihre Quell-Information auf die ihnen zugeordneten lokalen Exemplare übertragen. Diese wiederum könnten die Information an ihre Exemplare durch die Hierarchie weiterreichen. Es entstünden Multi-Source-Exemplare. Aber in einem solchen Modell kann die Wahrung des Clean-Source-Constraints zu einem schlechten Clustering-Ergebnis führen. Eine schlechte Entscheidung in einem

niedrigen Zweig der Hierarchie, bei der ein Datenpunkt einer „sauberen“ Quelle einem lokalen Exemplar mit relativ geringer Ähnlichkeit zugeordnet wurde, kann die Vereinigung zweier Zweige auf einer höheren Hierarchieebene blockieren. Viele Datenpunkte, die eigentlich dem gleichen Cluster angehören sollten, würden aufgrund des einen Punktes voneinander getrennt.

Eine bessere Methode, um die Clean-Source-Konsistenz zu wahren, ist die Modifikation des letzten Schrittes von HAP, also der Exemplarzuordnung. Dieser Ansatz erhält die Bezeichnung MSCD-HAP. Datenpunkte aus duplikatbehafteten Quellen können weiterhin ihr globales Exemplar anhand der größten Ähnlichkeit wählen. Für jede duplikatfreie Datenquelle muss die Exemplarzuordnung getrennt erfolgen. Dazu verwendet MSCD-HAP ein Verfahren zum Lösen gewichteter Zuordnungsprobleme, wie beispielsweise den Kuhn-Munkres-Algorithmus [26, 29] (auch *Ungarische Methode* genannt). Bei gegebenen Ähnlichkeiten zwischen Datenpunkten und globalen Exemplaren findet der Kuhn-Munkres-Algorithmus eine optimale Lösung, in der jeder Datenpunkt einem anderen Exemplar zugeordnet ist. Dabei maximiert das Verfahren die Summe der Ähnlichkeiten zwischen den Datenpunkten und den ihnen zugeordneten Exemplaren. Falls die Anzahl der Datenpunkte einer „sauberen“ Quelle jene der globalen Exemplare übersteigt, bilden die übrigen Punkte jeweils ein Singleton-Cluster. Das Entfernen der Ähnlichkeiten zwischen Datenpunkten der gleichen duplikatfreien Quelle, welches eine Voraussetzung für MSCD-AP ist (vgl. Abschnitt 4.2.1), garantiert auch in MSCD-HAP die Clean-Source-Konsistenz. Falls ein globales Exemplar aus einer duplikatfreien Quelle stammt, kann der Kuhn-Munkres-Algorithmus diesem somit keinen Punkt der gleichen Quelle zuordnen.

Die Ungarische Methode hat eine Komplexität von $\mathcal{O}(mn^2)$ für eine $m \times n$ Kostenmatrix [9, S. 822] mit n globalen Exemplaren und m Datenpunkten einer „sauberen“ Quelle. Die Komplexität übersteigt zwar jene von AP, doch die Exemplarzuordnung erfolgt jeweils nur auf einer sehr kleinen Teilmenge der eigentlichen Datensammlung ($m, n \ll N$). Folglich ist die Kombination aus HAP, MSCD-AP und dem Kuhn-Munkres-Algorithmus dennoch deutlich besser für das Clustering großer Datenmengen geeignet als MSCD-AP.

Da die Clean-Source-Konsistenz in MSCD-HAP durch die Exemplarzuordnung im letzten Schritt sichergestellt wird, könnte die Bestimmung der lokalen Exemplare auf den einzelnen Partitionen auch mittels AP anstelle von MSCD-AP erfolgen. Abschnitt 6.6 untersucht, ob die frühzeitige Berücksichtigung der Quellen auf den Partitionen durch MSCD-AP einen Vorteil einbringt.

4.3.2 Sparse MSCD-AP

Das Entity-Matching eines ER-Prozesses erzeugt im Allgemeinen einen Similarity-Graphen mit einer geringen Kantendichte bzw. eine dünnbesetzte Similarity-Matrix. Die Match-Funktion könnte die Kanten im einfachsten Fall durch einen Schwellwert bezüglich der Ähnlichkeit der verbundenen Knoten filtern. Je größer dieser Schwellwert eingestellt ist, desto geringer fällt die Kantendichte des Similarity-Graphen aus. Wie bereits in Abschnitt 3.4.3 erwähnt, ist AP dafür geeignet, sich eine geringe Kantendichte zunutze zu machen, um Nachrichten einzusparen. Bei Einsparung der nicht benötigten Nachrichten hat jede AP-Iteration eine Komplexität von $\mathcal{O}(Nk\,log\,N)$ [15, S. 14]. Dabei steht N für die Anzahl der Datenpunkte und k für die durchschnittliche Konnektivität des Similarity-Graphen. In Algorithmus 4.2.1 lässt sich erkennen, dass das Clean-Source-Constraint den Rechenaufwand von MSCD-AP gegenüber AP nur um einen konstanten Faktor erhöht. Die Komplexität von MSCD-AP entspricht also der seines Basisverfahrens.

Durch die Kantenfilterung in der Match-Funktion des ER-Prozesses ist es möglich, dass getrennte Teilgraphen entstehen. Diese lassen sich durch eine Connected-Components-Analyse identifizieren und anschließend einzeln clustern. Eine stärkere Filterung in der Match-Funktion erhöht die Anzahl und verringert die Größe der Teilgraphen. Damit ist es möglich, MSCD-AP direkt für das Clustering in einem ER-Prozess mit größeren Datenmengen einzusetzen. Der Informationsverlust durch die stärkere Filterung fällt geringer aus als jener der zufälligen Partitionierung im zuvor vorgestellten Ansatz MSCD-HAP. Anstatt durch Zufall erfolgt die Informationsreduktion in der Match-Funktion strukturiert, so dass ungewissere Kanten (z.B. solche mit geringer Datenpunktähnlichkeit) vor gewisseren entfernt werden. Bei zu großen Teilgraphen steigt der Rechenaufwand jedoch trotz eingesparter Nachrichten stark. MSCD-HAP hat also einen Vorteil in der Skalierbarkeit, Sparse MSCD-AP hingegen in der Clustering-Qualität.

Kapitel 5

Prototypische Implementierung

In Kapitel 4 wurde mit MSCD-AP eine Erweiterung für AP entworfen, die ein spezialisiertes Clustering für Multi-Source Clean-Dirty ER ermöglicht. Die beiden Ansätze MSCD-HAP und Sparse MSCD-AP sollen die Anwendung des MSCD-Clusterings auf große Datenmengen ermöglichen. Dieses Kapitel beschäftigt sich mit der prototypischen Umsetzung der skalierbaren Ansätze als Implementierungen für Apache Flink. Die Implementierungen wurden als Clustering-Module für das ER-System FAMER entwickelt. Damit sind sie mit den anderen in FAMER integrierten Clustering-Verfahren vergleichbar und können in einem vollständigen ER-Prozess eingesetzt und evaluiert werden. Der Quellcode ist auf der beiligenden DVD und auf dem öffentlichem GitLab-Repository[1] des FAMER-Projekts einsehbar.

Die Implementierungen unterscheiden sich grundlegend in der Herangehensweise bei der Umsetzung mit Apache Flink. Dennoch repräsentieren sie das gleiche Konzept und weisen Gemeinsamkeiten bei gewissen Details auf. Zunächst erfolgt eine Erläuterung dieser Gemeinsamkeiten in Abschnitt 5.1. Anschließend werden in den Abschnitten 5.2 und 5.3 zwei verschiedene Implementierungen für Sparse MSCD-AP vorgestellt. Aufgrund von Laufzeitproblemen der Umsetzung von Sparse MSCD-AP in Flink wurde das Verfahren einmal mit der grundlegenden DataSet-API und ein weiteres Mal unter Verwendung der Bibliothek Gelly implementiert (vgl. Abschnitt 3.2). Der letzte Abschnitt dieses Kapitels beschreibt die Umsetzung von MSCD-HAP.

[1] `https://git.informatik.uni-leipzig.de/dbs/FAMER/-/tree/master/famer-clustering/src/main/java/org/gradoop/famer/clustering/parallelClustering/affinityPropagation`

5.1 Gemeinsamkeiten

AP hat Schwierigkeiten zu konvergieren, wenn verschiedene Lösungen die Energiefunktion ähnlich gut optimieren (vgl. Abschnitt 3.4.3). Es kommt zu Oszillationen, die sich auch mittels Dämpfung und Rauschen nicht vollständig vermeiden lassen. Das in Abschnitt 3.5 vorgestellte Adaptive AP ermöglicht es, Oszillationen im laufenden AP-Prozess zu „entkommen“. Für die Implementierungen von MSCD-AP musste eine Entscheidung getroffen werden, wie mit dieser Problematik umzugehen ist. Die vollständige Umsetzung von Adaptive AP, besonders die des komplexeren *p-scannings*, ist für diese erste prototypische Machbarkeitsprüfung nicht erforderlich. Nach erfolgreicher Evaluation des Konzepts und einer Entscheidung zur Aufnahme des Algorithmus in FAMER, können zukünftig solche Verbesserungen integriert werden.

Für die Prototypen fiel die Entscheidung zugunsten einer einfacheren Parameter-Adaption aus. MSCD-AP arbeitet entsprechend des traditionellen AP für eine feste Anzahl von Iterationen. Ist das Verfahren in dieser Zeit nicht konvergiert, eventuell aufgrund von Oszillationen, erfolgt eine Parameteradaption „von außen“. Das bedeutet, die Parameter werden in Anlehnung an Adaptive AP angepasst, aber der Message-Passing Vorgang wird für die aktualisierten Parameter neu gestartet. Da sich in Experimenten zeigte, dass die Modifikation der Preference p schneller zu einer konvergierenden Lösung führt als die des Dämpfungsfaktors, erfolgt zunächst eine Anpassung von p. Der Parameter wird zu Beginn um eine konfigurierbare Schrittweite reduziert, wodurch sich die Anzahl der resultierenden Cluster verringert. Dies entspricht dem Vorgehen in Adaptive AP und folgt Beobachtungen bei der Anwendung von AP auf den Testdatensätzen von Kapitel 6. Bei diesen tendierte der Algorithmus dazu, zu viele Cluster zu erzeugen. Ist p nicht weiter reduzierbar, ohne einen Wert von null zu unterschreiten, wird es vom ursprünglichen Wert aus erhöht. Der Wert steigert sich jeweils um die gewünschte Schrittweite, bis dies nicht mehr möglich ist, ohne einen Wert von eins zu überschreiten. Hat MSCD-AP alle möglichen Werte von p erfolglos versucht, erhöht es den Dämpfungsfaktor λ um eine ebenfalls konfigurierbare Schrittweite. Für den neuen Wert von λ testet MSCD-AP wieder alle möglichen Werte von p, beginnend vom ursprünglichen Wert. Dieser Vorgang wiederholt sich, bis der Algorithmus für eine Parameterkonfiguration konvergiert oder bis sich λ nicht weiter erhöhen lässt, ohne den Wert von eins zu übersteigen. In dem Fall bricht das Verfahren ab, ohne eine Lösung gefunden zu haben.

In Abschnitt 4.3.2 wurde erläutert, dass im ER-Prozess durch die Kantenfilterung auf dem Similarity-Graphen einzelne Teilgraphen entstehen. Jeder Teilgraph lässt sich als ein eigenes Clustering-Problem betrachten. Daher wurde für alle Imple-

mentierungen die Entscheidung getroffen, Teilgraphen unabhängig voneinanander zu clustern, wobei die Verarbeitung im verteilten System Apache Flink parallel erfolgt. Zur Feststellung der Teilgraphen erfolgt zunächst eine Connected-Components-Analyse. Für jede Connected Component wird unabhängig geprüft, ob eine Lösung gefunden wurde oder ob eine Parameteradaption und ein Neustart des AP-Prozesses für den Teilgraphen erforderlich sind. Wenn MSCD-AP eine Lösung für eine Connected Component gefunden hat, wird die Verarbeitung für die Komponente eingestellt. Dadurch reduziert sich schrittweise die zu verarbeitende Datenmenge.

Bei der Implementierung von AP und MSCD-AP wurde festgestellt, dass die Binärmatrix $\mathbb{B}$ eines konvergierten AP-Prozesses nicht immer valide ist. Diese Beobachtung bestätigte sich bei Experimenten mit der Implementierung von AP im *scikit-learn* [33]. Tabelle 5.1.1 zeigt dies anhand eines Beispiels. AP konvergiert für die abgebildete Similarity-Matrix $\mathbb{S}$ und ermittelt die invalide Lösung $\mathbb{B}$, welche das 1-of-N-Constraint in der grün markierten Zeile missachtet. Die Summe aus α- und β-Nachricht ist in keiner der Zellen positiv. Eine Verletzung des 1-of-N-Constraints ist der am häufigsten beobachtete Fall eines invaliden Ergebnisses. Ein Versuch das Problem zu umgehen wäre es, b_{ij} für die größte Summe $\alpha_{ij}+\beta_{ij}$ der Zeile unabhängig vom Vorzeichen auf eins zu setzen. Dies führt jedoch zu einer häufigeren Verletzung des Exemplar-Consistency-Constraints.

Die Matlab-Implementierung von Frey und Dueck in [13, S. 2] sowie die Implementierung im *scikit-learn* umgehen das Problem. Sie identifizieren ausschließlich die Exemplare in $\mathbb{B}$ und ordnen die übrigen Datenpunkte, anhand der größten Ähnlichkeit in $\mathbb{S}$, diesen Exemplaren zu. Das ermöglicht die Erzeugung eines validen Clusterings, auch wenn die Binärmatrix invalid ist. Dieser Ansatz lässt sich allerdings nicht auf MSCD-AP übertragen, da dies die Separation der Punkte aus sauberen Quellen zunichtemachen würde. Anstatt dessen prüfen die Implementierungen von MSCD-AP die Validität von $\mathbb{B}$ bezüglich aller drei Constraints. Ist eines der Constraints verletzt, verfährt MSCD-AP so, als ob es nicht konvergiert sei. Die Lösung wird verworfen, die Parameter adaptiert und der Clustering-Prozess beginnt von

s_{ij}	0	1	2	3	4
0	0,21	0,24	0,11	0,20	0,21
1	0,24	0,21	0,75	0,79	0,00
2	0,11	0,75	0,21	0,79	0,07
3	0,20	0,79	0,79	0,21	0,10
4	0,21	0,00	0,07	0,10	0,21

(a) Similarity-Matrix $\mathbb{S}$

b_{ij}	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	**1**	0
2	0	0	0	**1**	0
3	0	0	0	**1**	0
4	0	0	0	0	**1**

(b) Binärmatrix $\mathbb{B}$

Tabelle 5.1.1: AP-Prozess mit invalider Lösung

Neuem. Damit ist die in den Anforderungen (vgl. Abschnitt 4.1) definierte strikte Trennung der Datenpunkte aus sauberen Quellen garantiert.

Die Cluster-Anzahl und damit auch die Qualität der Lösung sind stark vom Preference-Parameter p abhängig. Mit Hilfe von Parameter-Tuning lässt sich das Ergebnis des Algorithmus verbessern. Die in den nächsten Abschnitten vorgestellten Implementierungen bieten alle die gleichen Konfigurationsparameter, die im Folgenden erläutert werden.

Preference: Die Selbstähnlichkeit eines Datenpunktes legt fest, mit welcher Wahrscheinlichkeit ein Punkt zum Exemplar wird. Sie ist mit Hilfe von drei verschiedenen Parametern definierbar. Diese lassen sich jeweils separat für Datenpunkte aus „sauberen“ (CS) und „schmutzigen“ (DS) Quellen definieren. Dabei ist eine der drei folgenden Möglichkeiten auszuwählen:

Die Selbstähnlichkeit eines Datenpunktes, die festlegt mit welcher Wahrscheinlichkeit ein Punkt zum Exemplar wird, lässt sich über drei verschiedene Parameter definieren. Diese lassen sich jeweils separat für Datenpunkte aus „sauberen“ (CS) und „schmutzigen“ (DS) Quellen definieren. Dabei ist eine der drei folgenden Möglichkeiten auszuwählen:

- **Preference-Percentile (CS/DS):** Die Preference wird anhand eines Perzentils für alle Ähnlichkeiten des Teilgraphen bestimmt. Bei einem Wert von 40 entspricht dies beispielsweise jenem Wert, der von 60% der Ähnlichkeitswerte in $\mathbb{S}$ übertroffen wird. Dies ermöglicht eine Definition der Preference, die abhängig von den Eingabedaten ist. Frey und Dueck empfehlen die Verwendung des Median der Ähnlichkeitswerte (50%-Perzentil) für eine moderate Cluster-Anzahl [12, S. 972].

- **Minimum-Similarity (CS/DS):** Eine andere Möglichkeit, die Preference abhängig von den Eingabedaten zu definieren, ist das Minimum der Ähnlichkeitswerte. Damit ermittelt (MSCD-)AP eine Lösung mit geringer Cluster-Anzahl [12, S. 972]. Da das P-te Perzentil mit $0 < P < 100$ definiert ist [8, S. 26], lässt sich das Minimum nicht über ein Perzentil bestimmen und erfordert einen separaten Parameter.

- **Fix-Value (CS/DS):** Die dritte Möglichkeit zur Einstellung der Preference ist ein fester Wert. Dieser gilt für alle Teilgraphen und ist unabhängig von den Eingabedaten. Der *Fix-Value* ermöglicht die Unterschreitung des Minimums der Ähnlichkeitswerte und somit eine noch geringere Cluster-Anzahl im Ergebnis.

Preference-Adaption-Step: Dieser Parameter definiert die Schrittweite, um die MSCD-AP den Preference-Wert adaptiert, falls es mit der aktuellen Konfiguration nicht konvergieren konnte oder eine invalide Lösung gefunden hat.

Duplikatfreie Quellen: Die MSCD-AP Implementierungen ermöglichen die Festlegung der duplikatfreien Quellen über drei verschiedene Parameter. Wenn keine „sauberen“ Quellen definiert sind, arbeitet jeder der Prototypen entsprechend des Basisalgorithmus AP.

- **All Sources Clean:** Dies ist ein boolscher Parameter, über den sich definieren lässt, ob alle Quellen der Eingabedaten duplikatfrei sind. In dem Fall arbeitet MSCD-AP als Clustering-Algorithmus für MS-Clean-ER.
- **Source-Dirty-Vertex-Property:** Die zweite Möglichkeit ist die Definition des Namens einer Knoten-Eigenschaft, die über einen boolschen Wert angibt, ob die Quelle des Datenpunktes duplikatfrei ist.
- **Clean Sources:** Als dritte Variante lassen sich die „sauberen“ Quellen über eine Liste von Quellnamen festlegen.

Noise-Decimal-Place (*noiseLevel*): Das Rauschen wird in allen Implementierungen durch einen additiven, normalverteilten Zufallswert erzeugt. Der Parameter definiert die Stärke des Rauschens und orientiert sich grob an der Nachkommastelle der Datenpunktähnlichkeiten, auf die das Rauschen addiert wird. Ein gausscher Zufallswert r addiert sich auf jeden Ähnlichkeitswert mit $r \cdot 10^{-1 \cdot (noiseLevel\ -1)}$. Eine stark vom Erwartungswert abweichende Zufallszahl kann auch die Nachkommastelle $noiseLevel - 1$ verändern.

Damping-Factor: Der Dämpfungsfaktor λ definiert den Anteil des Nachrichtenwertes aus der vorangegangenen Iteration, der in den Wert der aktuellen Iteration mit einfließt. Dies soll Oszillationen vermeiden.

Damping-Adaption-Step: Ähnlich dem *Preference-Adaption-Step* definiert dieser Parameter die Schrittweite, um die MSCD-AP den Dämpfungsfaktor erhöht, falls es erfolglos alle möglichen Preference-Werte mit der aktuellen Konfiguration getestet hat.

Convergence-Iterations: Dieser Parameter bestimmt die Anzahl der Iterationen, in der die Exemplare eines Teilgraphen unverändert bleiben müssen, damit MSCD-AP konvergiert und die Lösung akzeptiert.

All-Equal-Similarity-Threshold: Wenn alle Ähnlichkeitswerte in einem Teilgraphen identisch sind, hat (MSCD-)AP Schwierigkeiten zu konvergieren. Das Verfahren kann dann nur schwer einen Datenpunkt auswählen, der das Cluster repräsentieren soll. Deshalb lässt sich ein Schwellwert für den gemeinsamen Ähnlichkeitswert bestimmen, bei dessen Unterschreitung jeder Datenpunkt ein Singleton bildet. Bei Überschreitung gruppiert AP alle Punkte in ein gemeinsames Cluster. MSCD-AP tut es diesem gleich, nur dass es Datenpunkte einer gemeinsamen duplikatfreien Quelle separiert und dabei die Cluster-Anzahl minimal hält. (MSCD-)AP startet für den Teilgraphen kein iteratives Message-Passing, sondern gibt direkt die Lösung aus.

5.2 Sparse MSCD-AP mit der Flink DatSet-API

Die erste prototypische Flink-Implementierung setzt Sparse MSCD-AP mit Hilfe der in Abschnitt 3.2.1 vorgestellten DataSet-API um. Abbildung 5.2.1 zeigt die grundlegende Idee hinter der Implementierung für das Einführungsbeispiel aus Abbildung 2.2.1. Die im iterativen Message-Passing von MSCD-AP ausgetauschten Nachrichten werden auf dünnbesetzte Matrizen abgebildet. Jeder Nachrichtentyp erhält eine eigene Matrix. Die Zellen ij der Matrizen sind nur an den Stellen besetzt, an denen der Similarity-Graph eine (bidirektionale) Kante zwischen den Knoten i und j aufweist sowie auf der Diagonalen, um die Nachrichten eines Knotens zu sich selbst zu repräsentieren. Alle besetzten Zellen der verschiedenen Matrizen, welche die gleiche Zeile und Spalte aufweisen, werden auf eine gemeinsame sogenannte Multimatrix-Zelle abgebildet. Dabei handelt es sich um ein Objekt, welches Attribute für die Zeilen- und Spalten-ID sowie für die Nachrichtentypen und andere Zwischenrechnungen enthält.

Rechts auf der Abbildung ist mit *ApMultiMatrixCell* die Klasse der Multimatrix-Zelle in UML-Notation dargestellt. Für jede besetzte Zelle wird ein eigenes Objekt erzeugt, was in der Abbildung durch die farbliche Kennzeichnung visualisiert ist. Die Nachrichten β und γ sind nur temporär für die Berechnung von η und θ relevant und müssen nicht permanent in den Multimatrix-Zellen vorgehalten werden. Anstatt dessen nutzt die Implementierung dafür ressourcenschonende temporäre Felder, welche auch für weitere Nebenrechnungen zum Einsatz kommen. Diese grundlegende Idee bringt alle notwendigen Informationen für die Implementierung von MSCD-AP in ein für die DataSet-API umsetzbares Format.

Die Multimatrix-Zelle ist ein POJO, welches im Sinne des Flink-Programmiermodells als Datentyp eines DataSets dient. Für verschiedene Berechnungen können

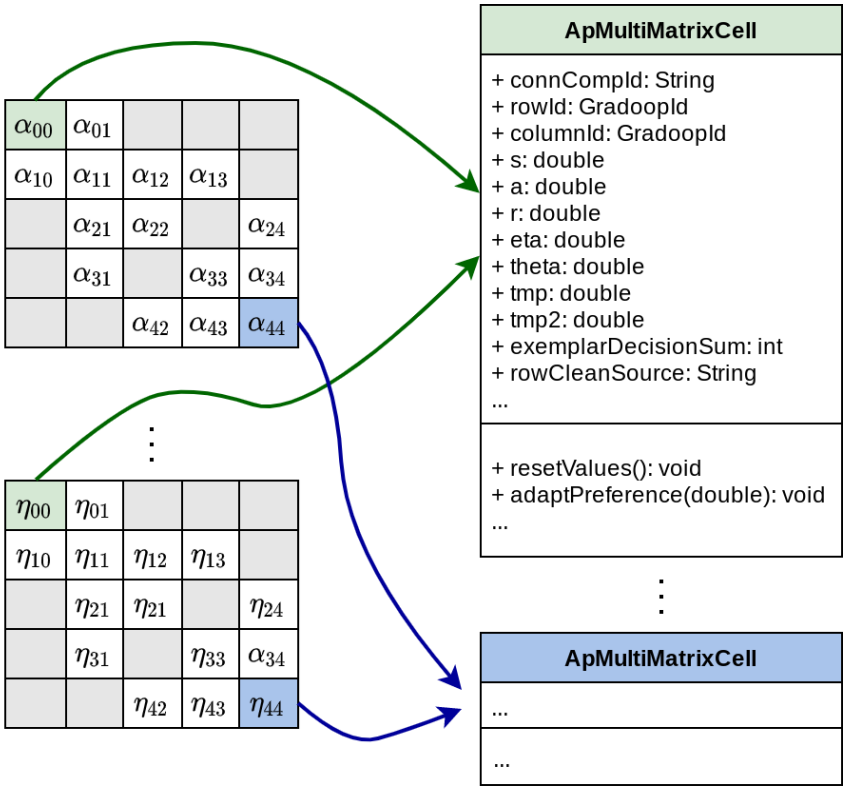


Abbildung 5.2.1: Zuordnung der dünnbesetzten Matrizen zu Multimatrix-Zellen

aus den Attributen Schlüssel gebildet werden. Aus der algorithmischen Betrachtung von MSCD-AP in Abschnitt 4.2.3 geht hervor, dass die Berechnungen der einzelnen Nachrichtenwerte ausschließlich auf Zeilen- und Spaltenoperationen auf den Matrizen sowie auf Matrixadditionen basiert. Eine Matrixaddition lässt sich durch eine einfache Map-Tranformation umsetzen, da die Werte der Matrizen aus den gleichen Zellen in derselben Multimatrix-Zelle vorhanden sind. Zeilen- und Spaltenoperationen können mit Hilfe einer Kombination aus GroupBy-, Reduce- und Join-Transformation ausgeführt werden. GroupBy gruppiert zunächst alle Zellen der gleichen Zeile anhand der *rowId* (bzw. der gleichen Spalte mittels der *columnId*). Reduce führt anschließend die gewünschte Berechnung auf der Gruppe aus. Es entsteht ein reduziertes DataSet, welches eine Multimatrix-Zelle für jede Zeile (bzw. Spalte) enthält. Das Ergebnis muss anschließend mit einem Join über die *rowId* (bzw. die *columnId*) auf die Multimatrix-Zellen des vollständigen DataSets übertragen werden.

Die Implementierung setzt das iterative Message-Passing von MSCD-AP mit Hilfe einer *DeltaIteration* um. Diese arbeitet mit einem *WorkingSet* und einem *SolutionSet*. Das WorkingSet enthält alle Multimatrix-Zellen der Teilgraphen, für die das Clustering in der aktuellen Iteration noch berechnet wird. Das SolutionSet enthält

alle Zellen der Teilgraphen, für die MSCD-AP bereits eine Lösung gefunden hat. Diese müssen bei den Operationen des Message-Passing-Vorgangs nicht mehr berücksichtigt werden. Ob das Verfahren für einen Teilgraphen eine Lösung gefunden hat, lässt sich am Ende jeder Iteration mit Hilfe der Felder *connCompId* und *exemplarDecisionSum* (vgl. Abb. 5.2.1) feststellen. Letzteres speichert für diagonale Zellen die Summe der vorangegangenen, aufeinanderfolgenden Iterationen, in denen sich die Entscheidung, ob es sich bei dem Datenpunkt der Zelle um ein Exemplar handelt, nicht mehr verändert hat. Ist der Datenpunkt ein Exemplar, dann ist die Summe positiv. Ansonsten ist sie negativ. Durch ein *GroupBy* auf die Connected-Component-ID kann nun eine Reduce-Funktion für alle Zellen eines Teilgraphens prüfen, ob sich die Exemplare innerhalb der letzten, durch den Parameter *convergenceIter* definierten Iterationen, verändert haben. Ist dies nicht der Fall, gilt der Teilgraph als konvergiert und seine Zellen werden zum SolutionSet hinzugefügt. Falls ein Teilgraph in der vorgegebenen Anzahl von Iterationen keine Lösung findet oder für eine Lösung konvergiert, die eines der Constraints verletzt, dann werden die Parameter entsprechend adaptiert und das Message-Passing von Neuem begonnen (*adaptPreference, resetValues*, vgl. Abb. 5.2.1).

Bei Experimenten für die Evaluation im Kapitel 6 stellte sich heraus, dass diese Implementierung von Sparse MSCD-AP sehr lange Laufzeiten benötigt. Dies belegt Abschnitt 6.3. Aufgrund dessen wurde eine alternative Implementierung des Algorithmus mit Hilfe der Bibliothek Gelly entwickelt, welche der folgende Abschnitt vorstellt.

5.3 Sparse MSCD-AP mit der Bibliothek Gelly

Die zweite prototypische Flink-Implementierung setzt Sparse MSCD-AP mit Hilfe der in Abschnitt 3.2.2 vorgestellten Bibliothek Gelly um. Gelly bietet eine API zur Verarbeitung von Graphen und dem Entwurf iterativer Message-Passing-Algorithmen. Somit ließe sich MSCD-AP damit direkt für den Nachrichtenaustausch auf einem Faktorgraphen umsetzen. Bei den großen Datenmengen im Big-Data-Umfeld wäre eine direkte Verwendung des Faktorgraphen jedoch sehr ressourcenintensiv und inperformant. Für N Datenpunkte und eine dichtbesetzte Similarity-Matrix erfordert ein Faktorgraph allein für das traditionelle AP eine Anzahl von $N^2 + 2N$ Knoten. Mit den Faktorknoten des Clean-Source-Constraints erhöht sich diese Zahl für MSCD-AP sogar noch.

Abbildung 5.3.1 zeigt einen sparsameren Ansatz. Ähnlich zur Implementierung im vorangegangenen Abschnitt werden auch hier zunächst sämtliche Zellen der dünn-

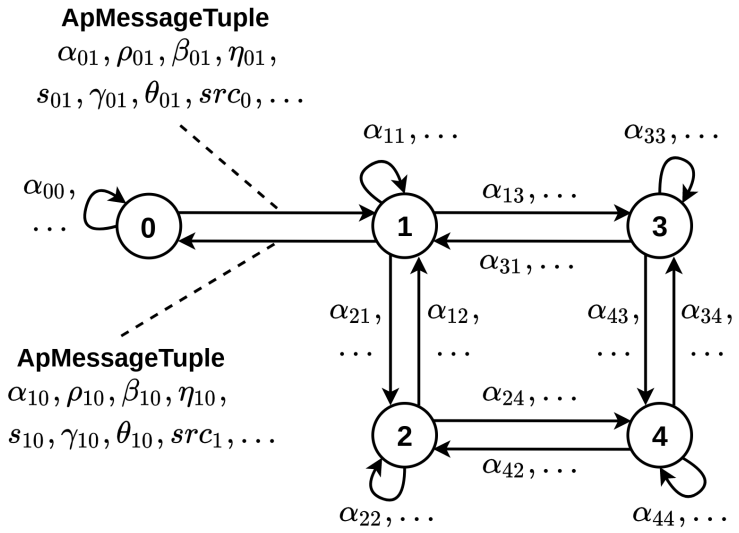


Abbildung 5.3.1: *Graphrepräsentation von Sparse MSCD-AP mit Gelly.* Abbildung der dünnbesetzen Nachrichtenmatrizen auf Nachrichtentupel.

besetzten Nachrichtenmatrizen auf ein zur Verarbeitung mit Flink geeignetes Konzept abgebildet. Im Fall von Gelly ist dies ein Graph. Der Darstellung liegt wieder das Einführungsbeispiel aus Abbildung 2.2.1 zugrunde. Die Eingabe für einen Clustering-Prozess in FAMER ist immer ein Similarity-Graph. Dieser lässt sich fast unverändert für die Verwendung in einem Message-Passing-Algorithmus mit Gelly einsetzen. Falls der Graph unidirektionale Kanten einsetzt, muss für jede Kante eine weitere in entgegengesetzter Richtung eingefügt werden. Außerdem erhält jeder Knoten eine zusätzliche Kante zu sich selbst, welche die Selbstähnlichkeit repräsentiert.

In einem so modifizierten Graphen kann jede Zelle ij einer dünnbesetzten Nachrichtenmatrix auf eine Nachricht abgebildet werden, welche ein Knoten i auf der Kante zu Knoten j sendet. Die ausgetauschten Nachrichten sind dabei Tupel (*ApMessageTuple*), die jeweils ein Feld für jeden Nachrichtentyp von MSCD-AP enthalten. Zur Speicherung der Knotenwerte setzt die Graphrepräsentation ebenfalls Tupel ein (*ApVertexValueTuple*, vgl. Abb. 5.3.2). Um die der Implementierung zugrundeliegenden Nachrichten von MSCD-AP auf dem Faktorgraph von jenen unterscheiden zu können, welche die Knoten beim Gelly Message-Passing austauschen, werden letztere im Folgenden als Nachrichtentupel bezeichnet.

MSCD-AP benötigt die Werte einiger Nachrichtentypen aus der vorangegangenen Iteration für die Berechnung von neuen Nachrichtenwerten (vgl. Algorithmus 4.2.1). Daher müssen alle Nachrichtentupel auf Abbildung 5.3.1 über den iterativen Message-Passing-Prozess in Gelly erhalten bleiben. Gelly ermöglicht es, die von einem Kno-

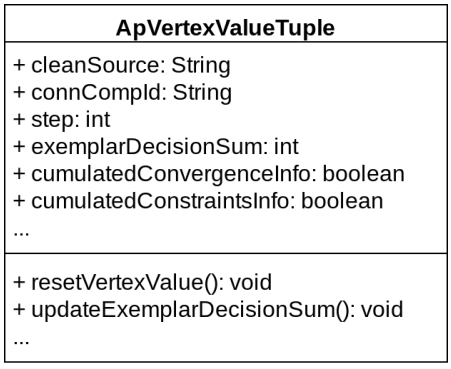


Abbildung 5.3.2: *Knotenrepräsentation in Sparse MSCD-AP mit Gelly.* Abbildung der Datenpunktinformationen auf Knotenwerte.

ten empfangenen Nachrichtentupel mit Hilfe eines *MessageCombiners* zu kombinieren, um die Kosten des Datenfluss-Prozesses der Gelly zugrunde liegenden DataSet-API zu reduzieren. Bei einer Kombination der Tupel gingen jedoch die einzelnen Nachrichtenwerte verloren. Die beiden Message-Passing-Modelle *Scatter-Gather* und *Gather-Sum-Apply* erfordern zwingend eine Kombination der Nachrichtentupel. Aus diesem Grund kommt für die Implementierung von MSCD-AP in Gelly nur das knotenzentrierte Modell in Frage.

Abbildung 5.3.3 zeigt, wie sich die für die Berechnung von MSCD-AP erforderlichen Zeilen- und Spaltenoperationen mit dem knotenzentrierten Message-Passing umsetzen lassen. Im ersten Superstep sendet jeder Knoten ein Nachrichtentupel auf jeder ausgehenden Kante zu seinen Nachbarknoten. Diese *ApMessageTupel* bilden alle dünnbesetzten Matrizen von MSCD-AP ab. In allen weiteren Supersteps antwortet jeder Knoten für ein empfangenes Tupel mit der Weiterleitung ebendieses. Dadurch empfängt ein Knoten im dritten Superstep wieder jene Tupel, welche er im ersten versendet hat. Dies setzt sich für die folgenden Supersteps fort. In abwechselnden Gelly-Iterationen erhält ein Knoten also einmal alle Nachrichtentupel einer Zeile und einmal alle Tupel einer Spalte und kann die notwendigen Rechenoperationen darauf ausführen. Die außerdem für MSCD-AP notwendigen Matrixadditionen können jederzeit ausgeführt werden, da alle dafür benötigten Informationen im gleichen Tupel enthalten sind.

Die Knoten manipulieren die einzelnen Werte der Tupel entsprechend der Berechnungen. Falls für eine Kalkulation ein Zwischenergebnis erforderlich ist, wird dieses im Knotenwert gespeichert. Der Knoten setzt die Rechnung nach zwei Supersteps mit Hilfe des gespeicherten Wertes fort. Im dazwischen liegenden Schritt kann gegebenenfalls eine andere Operation berechnet werden, wenn diese nicht auf dem aktuell berechneten Nachrichtenwert aufbaut. Mit Hilfe dieses Weiterleitungs-

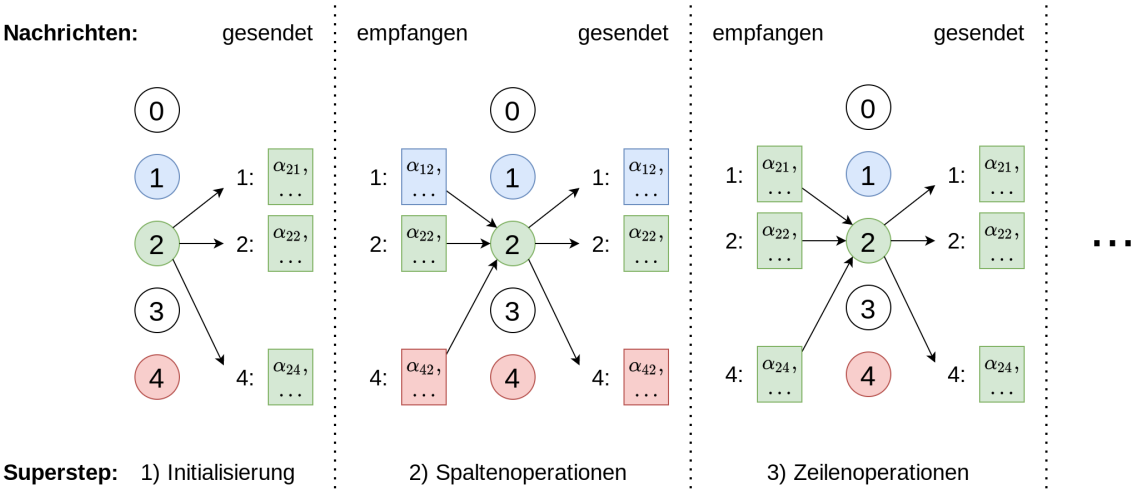


Abbildung 5.3.3: *Wechsel der Betrachtungsrichtung auf die Nachrichtenmatrizen in aufeinanderfolgenden Gelly-Iterationen.* Zur besseren Übersicht sind nur die von Knoten Nummer zwei gesendeten und empfangenen Nachrichten dargestellt. Die übrigen Knoten tauschen entsprechend Abbildung 5.3.1 ebenso Nachrichten aus.

tricks kann die Gelly-Implementierung eine MSCD-AP Iteration in acht Supersteps umsetzen, welche im Folgenden als Zwischenschritte bezeichnet werden.

Jeder Knoten speichert einen Zähler für den aktuell zu verarbeitenden Zwischenschritt im Knotenwert (vgl. Abb. 5.3.2: *step*). Je nach Zählerwert führt er unterschiedliche Operationen auf den Nachrichtentupeln aus, so dass Schritt für Schritt die Berechnungen einer MSCD-AP Iteration abgearbeitet werden. Nach Beendigung des achten Supersteps setzt sich der Zwischenschritt auf den Wert eins zurück und die nächste MSCD-AP Iteration beginnt. Ähnlich zu dem im vorangegangenen Abschnitt vorgestellten DataSet-API Prototyp, speichert jeder Knoten eine *exemplarDecisionSum*, anhand derer sich feststellen lässt, über wie viele Iterationen der Datenpunkt des Knotens unverändert ein Exemplar repräsentiert.

Sind genügend Iterationen vergangen, so dass MSCD-AP für einige Teilgraphen konvergieren könnte, prüft jeder Knoten seine *exemplarDecisionSum* und ob die Constraints in seiner Spalte und Zeile eingehalten werden. Damit ein Teilgraph erfolgreich konvergiert und das Message-Passing einstellen kann, müssen die Constraints an jedem Knoten eingehalten sein. Außerdem muss die *exemplarDecisionSum* aller Knoten den Parameterwert *Convergence-Iterations* (vgl. Abschnitt 5.1) erreichen oder übertreffen.

Alle Knoten des Teilgraphen müssen gemeinsam im gleichen Superstep das Message-Passing einstellen. Dazu müssen die Knoten sich untereinander darüber informieren, ob beide Prüfungen erfolgreich waren. Dies geschieht bereits nebenbei, während der Verarbeitung der MSCD-AP Schritte der nächsten Iteration und kann

zusätzliche Supersteps erfordern. Jeder Knoten teilt mit Hilfe des Nachrichtentupels seinen Nachbarn den Ausgang der Prüfungen mit. Die boolschen Prüfungsergebnisse werden UND-verknüpft, weitergeleitet und im Knotenwert gespeichert (*cumulatedConvergenceInfo, cumulatedConstraintsInfo*). Mit Hilfe eines Aggregators prüft der Algorithmus die Summe aller Knoten, für die sich das kollektive Prüfungsergebnis seines Teilgraphen nach Erhalt der letzten Nachricht geändert hat. Ist diese Summe null, dann verfügen alle Knoten in allen Teilgraphen über die gleiche Information.

Die Knoten der erfolgreich konvergierten Komponenten stellen das Message-Passing ein. Für konvergierte Teilgraphen mit verletzten Constraints oder erreichtem Iterationslimit erfolgt eine Parameteradaption und ein Neustart des MSCD-AP Prozesses. Die übrigen Knoten setzen das Message-Passing mit der nächsten Iteration fort. Somit benötigt dieser Prototyp ab dem Zeitpunkt, an dem das Verfahren konvergieren kann, $8+X$ Supersteps für eine MSCD-AP Iteration. X steht für die variable Anzahl zusätzlicher Schritte zum Propagieren der Prüfergebnisse. Eine sehr dünn besetzte Similarity-Matrix bringt zwar den Vorteil, dass viele Nachrichten zwischen den Knoten eingespart werden können. Allerdings kann sie auch zu einer großen Zahl zusätzlicher Supersteps X führen, denn ein Graph mit geringer Kantendichte benötigt mehr Schritte für das Propagieren der Prüfungsergebnisse.

Diese Variante von Sparse MSCD-AP ist, ebenso wie der im vorangegangenen Abschnitt vorgestellte Prototyp, relativ unperformant. Im Vergleich zu anderen Clustering-Algorithmen in FAMER benötigt sie eine deutlich längere Laufzeit. Dies belegt die Evaluation in den Abschnitten 6.3 und 6.5. Die Implementierung erfordert sehr viele Supersteps des Gelly Message-Passings, um einen vollständigen MSCD-AP Prozess auszuführen. Jeder der Supersteps ist relativ kostenintensiv, da keine Einsparungen durch Nachrichtenkombination mit einem *MessageCombiner* möglich sind. Der im nächsten Abschnitt vorgestellte Prototyp erreicht eine deutlich bessere Performance.

5.4 MSCD-HAP für Apache Flink

Die dritte prototypische Flink-Implementierung setzt MSCD-HAP mit Hilfe der DataSet-API um. Anders als bei dem in Abschnitt 5.2 vorgestellten DataSet-API Prototyp für Sparse MSCD-AP erfolgt die eigentliche Ausführung des Clustering-Algorithmus in dieser Implementierung nicht vollständig parallelisiert, sondern sequentiell innerhalb einer GroupReduce-Funktion. Abbildung 5.4.1 visualisiert den Arbeitsablauf der Implementierung von MSCD-HAP. Im ersten Schritt werden mit Hilfe der Connected-Components-Analyse der Bibliothek Gelly Teilgraphen identi-

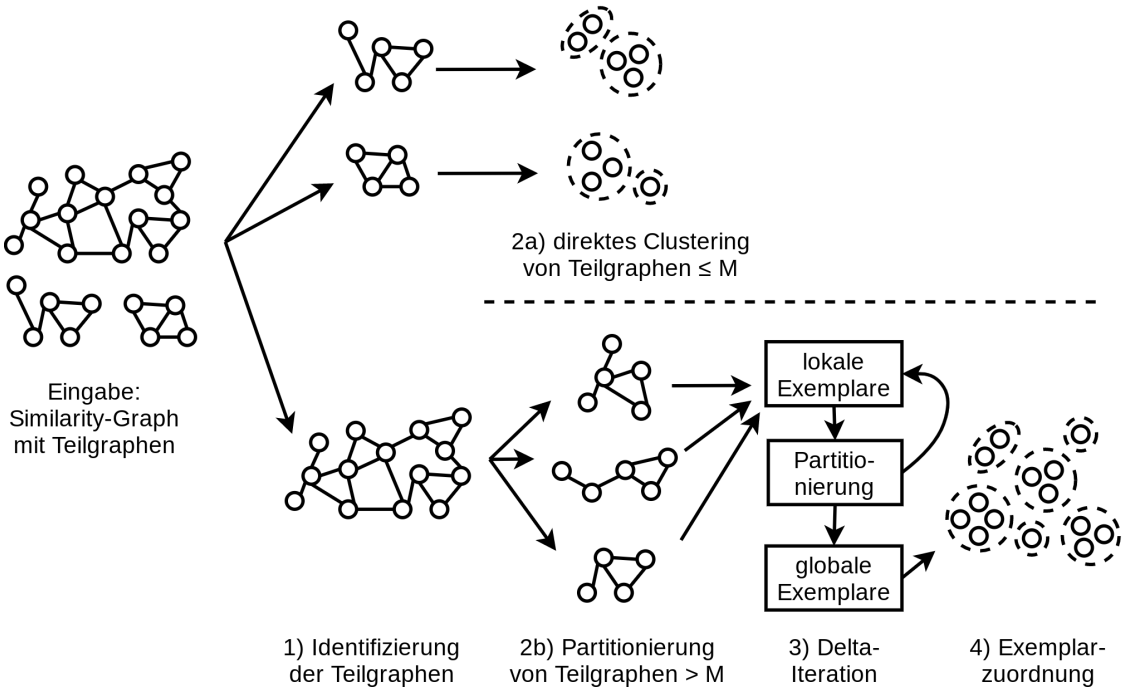


Abbildung 5.4.1: Vollständiger Arbeitsablauf von MSCD-HAP

fiziert und für jeden Datenpunkt als *connCompId* gespeichert. Anschließend prüft das Verfahren die Datenpunktanzahl der Teilgraphen. Überschreitet diese die per Parameter definierte maximale Partitionsgröße M, so erfolgt im Schritt $2b$ eine zufällige Partitionierung der Datenpunkte in gleich große Partitionen. Die Größe der Partitionen liegt so nahe wie möglich bei M, aber übersteigt dessen Wert nicht. Das Vorgehen für die partitionierten Komponenten wird später erläutert. Zunächst erfolgt eine Betrachtung des Schritts $2a$ für die Verarbeitung der Teilgraphen, deren Datenpunktanzahl M nicht überschreitet.

Zwei Joins der Knoten auf die Kanten bringen alle nötigen Informationen in ein gemeinsames DataSet. Dieses wird anhand der *connCompId* nach Teilgraphen gruppiert. Für jede Gruppe ermittelt eine GroupReduce-Funktion das Clustering-Ergebnis. Die Verarbeitung der einzelnen Gruppen erfolgt dabei parallel. Die GroupReduce-Funktion erzeugt zunächst eine Similarity-Matrix aus den Kantengewichten des eingegebenen Similarity-Teilgraphen. Anschließend ermittelt sie das Clustering für die Matrix mittels einer sequentiellen Implementierung von MSCD-AP. Diese basiert auf der AP-Implementierung von Smith in *clust4j* [42] und wurde um die in Abschnitt 5.1 beschriebenen Gemeinsamkeiten der Prototypen sowie um die Funktionalitäten von MSCD-AP erweitert. Die GroupReduce-Funktion speichert schließlich das Clustering-Ergebnis in den Knotenwerten des Teilgraphen und gibt ein DataSet der Knoten zurück. Dieses Vorgehen ist deutlich performanter als jenes der beiden in

den vorangegangenen Abschnitten beschriebenen Prototypen. Es erspart einen Großteil des Kommunikationsaufwands zwischen den Rechnerknoten des Flink-Clusters (vgl. Abschnitt 3.2). Das Clustering eines Teilgraphen erfolgt innerhalb eines einzigen Datenfluss-Transformationsschrittes auf einem einzelnen Task-Manager, wobei alle Teilgraphen parallel von den Task-Managern verarbeitet werden.

Abbildung 5.4.2 zeigt das Vorgehen für partitionierte Teilgraphen anhand eines Beispiels mit $M = 5$. Die Partitionierung erfolgt parallel für jeden der umfangreichen Teilgraphen auf einem eigenen Task-Manager sequentiell innerhalb einer GroupReduce-Funktion. Im dritten Schritt setzt eine Delta-Iteration die Ermittlung der Exemplare für die partitionierten Teilgraphen entsprechend des hierarchischen Teile-und-Herrsche-Verfahrens um. Jede Iteration entspricht dabei einer Hierarchieebene von MSCD-HAP. Zur Vermeidung endloser Wiederholungen in Ausnahmesituationen, bei denen sich die Exemplaranzahl durch das Clustering auf einer Hierarchieebene nicht reduziert, lässt sich eine maximale Hierarchietiefe definieren. In jeder Iteration werden die Partitionen des aktuellen *WorkingSets* anhand der Partitionsnummer und der *connCompId* gruppiert. Das Clustering jeder Gruppe erfolgt, analog zum oben beschriebenen Vorgehen für unpartitionierte Teilgraphen, sequentiell durch eine GroupReduce-Funktion. Die Task-Manager des Flink-Systems verarbeiten die einzelnen Partitionen parallel. Allerdings werden dabei nur die lokalen bzw. globalen Exemplare der Partition bestimmt. Die Exemplarzuordnung findet im Anschluss der Delta-Iteration statt. Die lokalen Exemplare aller Partitionen eines

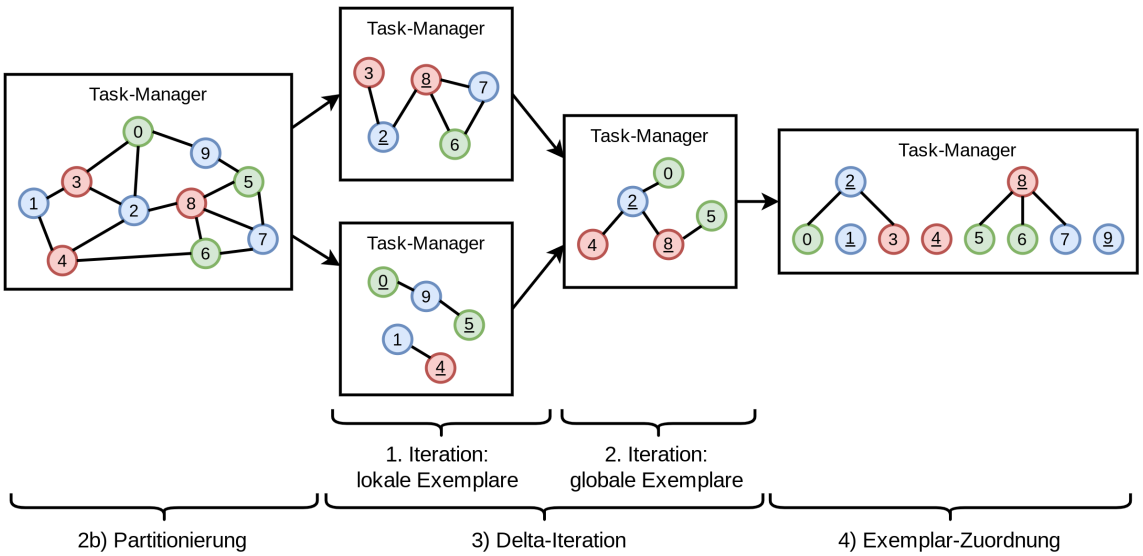


Abbildung 5.4.2: *Beispielhaftes Clustering eines partitionierten Teilgraphen in MSCD-HAP.* Die Farben kennzeichnen die jeweilige Datenquelle eines Knotens. Die rot und blau dargestellten Quellen liegen duplikatfrei vor und die grüne Quelle ist duplikatbehaftet.

Teilgraphen werden erneut partitioniert und bilden anschließend das WorkingSet der nächsten Iteration. Existiert für eine Connected-Component nur eine Partition, so stellt die aktuelle Iteration die höchste Hierarchieebene der Komponente dar. Die durch das sequentielle MSCD-AP ermittelten globalen Exemplare solcher Partitionen bilden das SolutionSet. Wenn das WorkingSet leer ist, weil alle Teilgraphen ihre höchste Hierarchieebene erreicht haben, beendet sich die Delta-Iteration. Im Beispiel von Abbildung 5.4.2 werden nur zwei Iterationen benötigt. Aus den fünf unterstrichen dargestellten lokalen Exemplaren der ersten Iterationen ergeben sich die Datenpunkte zwei und acht als globale Exemplare in der zweiten Iteration.

Im Anschluss an die Delta-Iteration erfolgt in Schritt vier die Zuordnung der Datenpunkte des partitionierten Teilgraphen zu den globalen Exemplaren. Dies geschieht nach einigen vorverarbeitenden Transformationen ebenfalls durch eine GroupReduce-Funktion, welche auf jeweils einem Task-Manager für jeden der ursprünglichen Teilgraphen sequentiell ausgeführt wird. Ein GroupBy gruppiert die Kanten zwischen Datenpunkten und Exemplaren nach *connCompId* und „sauberer“ Datenpunktquelle. Alle Punkte aus duplikatbehafteten Quellen bilden eine gemeinsame Gruppe. Sie werden jenem Exemplar zugewiesen, zu dem sie die größte Ähnlichkeit aufweisen. Jede Gruppe einer „sauberen“ Quelle durchläuft den Kuhn-Munkres-Algorithmus (Ungarische Methode) zur Exemplarzuordnung. Für diesen ruft die GroupReduce-Funktion die sequentielle Implementierung von Stern aus [43] auf.

Im Beispiel von Abbildung 5.4.2 lassen sich die Punkte null und drei dem Exemplar zwei zuordnen. Dem Exemplar acht werden die Punkte fünf, sechs und sieben zugeordnet. Weil die Punkte eins und neun über keine Kante zu einem der globalen Exemplare verfügen, müssen diese Singleton-Cluster bilden. Punkt vier bildet ebenfalls ein Singleton, da die Exemplarzuordnung mit der Ungarischen Methode dafür sorgt, dass er von dem aus der gleichen „sauberen“ Quelle stammenden Punkt drei separiert wird. Wäre die Ähnlichkeit von Punkt vier zu Punkt zwei größer als jene von Punkt drei, dann würde dieser anstatt dessen dem Exemplar zwei zugeordnet sein. Die Datenpunkte fünf und sechs stammen aus einer duplikatbehafteten Quelle und können daher beide in das gleiche Cluster gruppiert werden. Nach Abschluss der Exemplarzuordnung für alle Teilgraphen ist das Clustering mit MSCD-HAP vollendet.

Der in diesem Abschnitt vorgestellte Prototyp stellt die performanteste der drei Implementierungen von MSCD-AP dar. Durch den hierarchischen Partitionierungsansatz erzielt das Verfahren allerdings schlechtere Clustering-Ergebnisse bei großen Teilgraphen, weil Datenpunkte zufällig zu den Partitionen zugeordnet werden und

somit die Ähnlichkeiten zwischen Punkten aus unterschiedlichen Partitionen nicht mit in die Berechnung der lokalen Exemplare einfließt. Die verwendeten und modifizierten Code-Ausschnitte aus [42] und [43] sind unter der Apache 2.0 und der MIT-Lizenz veröffentlicht. Damit lassen sie sich problemlos im ebenfalls unter der Apache 2.0 Lizenz stehenden FAMER einsetzen. Im Quellcode wurden sämtliche urheberrechtlich relevanten Vermerke und Verweise auf den ursprünglichen Code vorgenommen.

Kapitel 6

Evaluation

Dieses Kapitel beschreibt die Evaluation der im vorangegangenen Kapitel vorgestellten prototypischen Implementierungen von MSCD-AP. Die Vorgehensweise orientiert sich an dem von Saeedi et al. in [37] durchgeführten Vergleich verschiedener Clustering-Algorithmen für Multi-Source-ER, in Rahmen dessen das FAMER-System entwickelt wurde. Ziel ist es, die Effektivität und Effizienz sowohl der neu implementierten als auch der bereits in Famer vorhandenen Clustering-Algorithmen für den Einsatz in MSCD-ER zu bestimmen und zu vergleichen. Dazu werden die Qualität der Ergebnisse, die Laufzeit und die Skalierbarkeit der Algorithmen gemessen und verglichen. Die Evaluation beantwortet hauptsächlich die folgenden Fragestellungen:

- Sind die entwickelten Prototypen für den Einsatz im Big-Data-Umfeld geeignet?
- Welchen Vorteil kann das spezialisierte MSCD-AP beim Einsatz in MSCD-ER gegenüber Algorithmen für MS-Clean-ER und MS-Dirty-ER erzielen?
- MS-Clean-ER ist ein Sonderfall von MSCD-ER, bei dem sämtliche Quellen duplikatfrei vorliegen. Wie gut ist MSCD-AP hierfür geeignet?
- Welchen Unterschied macht das neue Clean-Source Constraint bezüglich der Qualität des Clusterings aus? Wie unterscheiden sich also die Ergebnisse von MSCD-AP von denen des Basisalgorithmus AP?
- Wie wirkt sich die Partitionsgröße in MSCD-HAP auf die Laufzeit und die Ergebnisqualität aus?
- Bringt es einen Vorteil, MSCD-AP für die Bestimmung der lokalen und globalen Exemplare auf den Partitionen in MSCD-HAP einzusetzen oder genügt dafür der Einsatz von AP?

Die folgenden Abschnitte dieses Kapitels widmen sich der Beantwortung dieser Fragen. Abschnitt 6.1 präsentiert und erläutert zunächst die dafür verwendeten Datensammlungen. Im Anschluss stellt Abschnitt 6.2 die Evaluationsumgebung und die Konfiguration der Algorithmen für die Experimente vor. Abschnitt 6.3 beschäftigt sich mit der Frage, ob sämtliche Prototypen für den Einsatz im Big-Data-Umfeld geeignet sind und schränkt die Evaluation diesbezüglich ein. In den Abschnitten 6.4 und 6.5 erfolgt eine vergleichende Beurteilung der Ergebnisqualität, Laufzeit und Skalierbarkeit von einem der MSCD-AP Prototypen mit den in FAMER vorhandenen Clustering-Verfahren. Abschnitt 6.6 widmet sich einer detaillierten Betrachtung der wichtigsten Aspekte von MSCD-HAP und deren Einfluss auf die Effektivität und die Effizienz des Clusterings. Abschließend fasst Abschnitt 6.7 die Erkenntnisse dieses Kapitels anhand der eingangs gestellten Fragen zusammen.

6.1 Datensammlungen

Die praxisnahe, aussagekräftige Evaluation eines ER-Systems erfolgt am besten anhand von Testdaten aus der realen Welt. Für die Evaluation von MSCD-AP wurden zwei reale und zwei teilweise künstliche Datensammlungen aus den Domänen Digitalkameras (DS-C), Geographie (DS-G), Musik (DS-M) und Personen (DS-P) verwendet. Ihre wichtigsten Eigenschaften sind in Tabelle 6.1.1 aufgeführt. Das Raute-Symbol steht in sämtlichen Tabellen dieses Kapitels als Abkürzung für eine Anzahl. Die drei zuletzt genannten Datensammlungen enthalten ausschließlich duplikatfreie Quellen und wurden bereits mehrfach für die Evaluation von MS-Clean-ER eingesetzt [30, 37, 38, 39]. Da es sich bei MS-Clean-ER um einen Spezialfall von MSCD-ER handelt, sind die Datensammlungen auch für die Evaluation von MSCD-AP relevant. Sämtliche Datensammlungen und Evaluationsergebnisse sind auf der beiliegenden DVD angefügt.

Die kleinste Datensammlung ist DS-G. Sie enthält reale geographische Ortsbezeichnungen und Koordinaten aus vier verschiedenen Quellen (*DBpedia.org, Geonames.org, Freebase.com, NYTimes.com*). Das perfekte Ergebnis wurde durch eine manuelle Begutachtung der Daten festgestellt. DS-M basiert auf Informationen von Liedern aus der *MusicBrainz.org* Datenbank. Die größte Datensammlung stammt aus der Personen-Domäne und wurde in zwei Teilkorpora DS-P1 und DS-P2 untergliedert, um das Clustering verschiedener Datenmengen zu untersuchen. Sie basiert auf realen Personendatensätzen aus dem öffentlichen Wählerregister[1] des amerikanischen Bundesstaates North Carolina. Da die Daten in DS-M und DS-P aus einer

[1] `https://www.ncsbe.gov`

	Allgemeine Informationen				Perfektes Ergebnis	
	Domäne	Felder	#Beschr.	#Quellen	#Cluster	#Links
DS-C	Digitalkameras	heterogene Schlüssel-Wert-Paare	21.023	23	3.910	368.564
DS-G	Geographie	Bezeichnung, Längengrad, Breitengrad	3.054	4	820	4.391
DS-M	Musik	Künstler, Titel, Album, Jahr, Länge	19.375	5	10.000	16.250
DS-P1	Personen	Vorname, Nachname, Bezirk, PLZ	5.000.000	5	3.500.840	3.331.384
DS-P2			10.000.000	10	6.625.848	14.995.973

Tabelle 6.1.1: Datensammlungen für die Evaluation

einzelnen Quelle stammen, wurden künstlich Duplikate erzeugt, um weitere Quellen zu simulieren. In DS-M erfolgte die Duplikaterzeugung mit Hilfe des *DAPO* [22] Datengenerators. Die erzeugte Datensammlung enthält 50% Duplikate, die über zwei bis fünf verschiedene Quellen verteilt sind. Sie sind gegenüber den Originalen hochgradig abgewandelt, um eine möglichst große Herausforderung für ein ER-System darzustellen. Für die Duplikaterzeugung in der Personen-Domäne wurde das System *GeCo* [6] zur Datengenerierung und -korrumpierung eingesetzt. Die Korpora DS-P1 und DS-P2 bestehen aus fünf bzw. zehn Quellen, von denen jede jeweils eine Million Datensätze umfasst. 50% der Entitäten sind ohne Abwandlung über alle Quellen repliziert, 25% sind korrumpiert und in allen Quellen vertreten und die übrigen 25% sind ebenfalls abgewandelt, aber nur über eine Untermenge der Quellen verteilt. Bei der Erzeugung der abgewandelten Duplikate wurde eine moderate Korrumpierungsrate von 20% eingesetzt. Da die Duplikate in DS-M und DS-P automatisch erzeugt wurden, ist das perfekte Ergebnis (*Golden Truth*), welches von ER im bestmöglichen Fall erreicht werden könnte, bekannt. [37]

Im Gegensatz zu den anderen Datensammlungen wurde DS-C speziell für den Rahmen dieser Masterarbeit erstellt. Die Grundlage des Korpus bildet eine Sammlung von Produktspezifikationen, die für den *ACM SIGMOD 2020 Programming Contest*[2] als Vergleichsmaßstab verschiedener ER-Systeme diente. Es enthält ungefähr 30.000 Produktspezifikationen, die von 24 verschiedenen E-Commerce-Websites extrahiert wurden. Für den Programmierwettbewerb verunreinigte die *Database Research Group* der *Roma Tre University* die Kamerasammlung mit Spezifikationen sehr unterschiedlicher Produkttypen, wie beispielsweise von TV-Geräten oder Schuhen.

[2] `http://www.inf.uniroma3.it/db/sigmod2020contest/index.html`

	Blocking-Key	Ähnlichkeitsfunktionen	Gewicht	Match-Funktion
DS-G	PräfixLänge1 (Name)	sim1: JaroWinkler (Name) sim2: geographische Dist.(Längengrad, Breitengrad)	1 1	sim1 $\geq$ 0 UND sim2 $\leq$ 1358 km
DS-M	PräfixLänge1 (Titel)	sim1: Dice3 (Titel)	1	sim1 $\geq$ 0
DS-P	PräfixLänge3 (Vorname) + PräfixLänge3 (Nachname)	sim1: JaroWinkler (Vorname) sim2: JaroWinkler (Nachname) sim3: JaroWinkler (Bezirk) sim4: JaroWinkler (PLZ)	1 1 1 1	sim1 $\geq$ 0 UND sim2 $\geq$ 0.9 UND sim3 $\geq$ 0 UND sim4 $\geq$ 0
DS-C	PräfixLänge3 (Hersteller) + PräfixLänge100 (Modell)	sim1: maxListe (TruncateBegin20 (Modell-Liste)) sim2: maxListe (TruncateBegin20 (MPN-Liste)) sim3: maxListe (TruncateBegin20 (EAN-Liste)) sim4: Dice3 (Produktname) sim5: maxListe (3Gram (Modell-Liste)) sim6: NumMaxProz30 (Megapixel) sim7: NumMaxProz30 (Digitalzoom) sim8: NumMaxProz30 (optischer Zoom) sim9: NumMaxProz30 (Breite) sim10: NumMaxProz30 (Höhe) sim11: NumMaxProz30 (Gewicht) sim12: Jaccard3 (Productcode) sim13: Jaccard3 (Sensortyp) sim14: NumMaxProz30 (Preis) sim15: NumMaxProz30 (Auflösung X) sim16: NumMaxProz30 (Auflösung Y)	0 0 0 10 10 2 2 2 2 2 2 2 2 1 1 1	sim1 $\geq$ 1 ODER sim2 $\geq$ 1 ODER sim3 $\geq$ 1 ODER sim4 $\geq$ 0.8 ODER sim5 $\geq$ 0.6 ODER

Tabelle 6.1.2: Konfigurationen für das Blocking und Entity-Matching der verschiedenen Datensammlungen

Es sind jedoch auch Beschreibungen von Kamerazubehör wie Linsen, Akkus oder Taschen in der Datensammlung enthalten, welche schwieriger von den eigentlichen Kameraspezifikationen zu unterscheiden sind. DS-C soll zukünftig als Benchmark für MSCD-ER dienen und ausschließlich Beschreibungen von Digitalkameras enthalten. Die Quelle *alibaba.com* enthielt besonders viele Datensätze, bei denen es sich nicht um Kameras handelte, und wurde deshalb aus DS-C entfernt. Die übrigen Quellen ließen sich mittels Filterung von bestimmten Schlüsselwörtern wie „television“ oder „bag“ bereinigen. Für die Filterung konnte der Code der Wettbewerbsteilnehmer von der Universität Leipzig wiederverwendet werden. Nach Bereinigung und Quellausschluss ergibt sich aus der Datensammlung des SIGMOD Wettbewerbs das in Tabelle 6.1.1 aufgeführte DS-C. Als *Golden Truth* für die Evaluation der ER-Qualität dient das Ergebnis des Wettbewerbsgewinners, welches ein F-Maß (vgl. Abschnitt 6.4) von 99% erzielen konnte.

Tabelle 6.1.2 zeigt die Konfigurationen für das Blocking und das Entity-Matching, mit deren Hilfe Similarity-Graphen für die verschiedenen Datensammlungen erzeugt wurden. „PräfixLänge3“ bedeutet, dass die ersten drei Zeichen des aufgeführten At-

tributs in den Blocking-Key einfließen. Die Ähnlichkeit zweier Beschreibungen des gleichen Blocks ergibt sich aus der gewichteten Ähnlichkeit über die aufgeführten Ähnlichkeitsfunktionen. Dabei bedeutet ein Gewicht von null, dass die Funktion nicht in die Datenpunktähnlichkeit einfließt, sondern nur für die Match-Funktion relevant ist. „NumMaxProz30“ steht für eine numerische Ähnlichkeitsfunktion, die eine Ähnlichkeit von null ausgibt, wenn sich die beiden Zahlen um 30% oder mehr unterscheiden. Bei geringeren Unterschieden skaliert das Funktionsergebnis zwischen null und eins. „TruncateBegin20“ gibt eine Ähnlichkeit von eins aus, wenn die ersten 20 Zeichen zweier Zeichenketten identisch sind. Andernfalls ergibt die Funktion einen Wert von null. Für manche Attribute kommen verschiedene Werte in Frage, welche in einer Liste geführt werden. „MaxListe“ berechnet die Ähnlichkeiten aller möglichen Kombinationen der Listenelemente und gibt deren Maximum zurück. „Dice3“ und „Jaccard3“ stehen für eine Q-Gram Zeichenkettenähnlichkeit mit $Q = 3$, die mittels des Dice- bzw. des Jaccard-Koeffizienten bestimmt ist (vgl. Abschnitt 3.1.3). Die Jaro-Winkler-Funktion ist ein spezielles Ähnlichkeitsmaß für den Vergleich von Namen. Die Konfiguration für DS-G, DS-M und DS-P stammt von Saeedi et al. [38].

Für DS-C musste eine neue Blocking- und Matching-Konfiguration entworfen werden. Dies erforderte zunächst die Erzeugung eines gemeinsamen Schemas. Die von den Ähnlichkeitsfunktionen in Tabelle 6.1.2 verwendeten Attribute ließen sich mit Hilfe von Suchworten aus den heterogenen Schlüssel-Wert-Paaren extrahieren. Weil viele Beschreibungen nur über einen Bruchteil der Merkmale verfügen, war eine große Anzahl an Attributen nötig, damit verschiedene Beschreibungen eine hinreichende Attribut-Schnittmenge bilden konnten. Für einige Merkmale, wie beispielsweise die *Manufacturer Part Number* (MPN), konnten häufig mehrere Werte extrahiert und in einer Liste gespeichert werden. Die Match-Funktion von DS-C ist absichtlich so gestaltet, dass auch einige inkorrekte Kanten im Similarity-Graphen entstehen. Die Oder-Verknüpfung der ersten drei Ähnlichkeiten *sim1* bis *sim3* würde einen sehr genauen Ähnlichkeitsgraphen erzeugen. Das Hinzufügen von *sim4* und *sim5* schwächt die Genauigkeit etwas ab, damit die Clustering-Algorithmen auch Spielraum haben, das Ergebnis zu verbessern.

Die Evaluation von MSCD-ER soll auf Datensammlungen mit verschiedenen Anteilen von duplikatbehafteten und -freien Quellen erfolgen. Dafür wurden die Datenquellen in DS-C dedupliziert, so dass sich anschließend Teilkorpora mit verschiedenen Zusammenstellungen von „sauberen“ und „schmutzigen“ Quellen erzeugen ließen. Bei der Deduplikation dienten die Cluster der *Golden Truth* als Hilfsmittel. Aus jedem Cluster wurde nur eine Beschreibung pro Quelle in die deduplizierte Version der Datensammlung übernommen. Die Auswahl des besten Vertreters seiner

Datenquelle	ID	#Spez.	#dedup.
buy.net	1	358	244
cammarkt.com	2	198	94
buzzillions.com	3	832	630
cambuy.com.au	4	118	56
camerafarm.com.au	5	120	59
canon-europe.com	6	164	163
ebay.com	7	14.009	3.255
eglobalcentral.co.uk	8	190	75
flipkart.com	9	118	47
garricks.com.au	10	130	69
gosale.com	11	895	578
henrys.com	12	181	137
ilgs.net	13	102	64
mypriceindia.com	14	347	279
pcconnection.com	15	211	126
price-hunt.com	16	327	282
pricedekho.com	17	366	325
priceme.co.nz	18	740	475
shopbot.com.au	19	516	334
shopmania.in	20	630	556
ukdigitalcameras.co.uk	21	129	73
walmart.com	22	195	115
wexphotographic.com	23	147	87
Summe:		21.023	8.123

Tabelle 6.1.3: Übersicht der Quellen in DS-C über die Anzahl der Produktspezifikationen in originaler sowie in dedublizierter Ausführung

Name	%cln[1]	cln[2]	#cln[3]	#dirty[4]
DS-C0	0		0	21.023
DS-C26	26	1-6, 8-23	4.868	14.009
DS-C32	32	7	3.255	7.014
DS-C50	50	7, 18, 19,20, 22, 23	4.822	4.786
DS-C62A	62	1, 4, 6 7, 9, 11, 13, 15, 17, 19, 20	5.748	3.536
DS-C62B	62	2, 3, 5, 7, 8, 10, 12, 14, 16, 18, 21-23	5.630	3.478
DS-C80	80	1-12 14-18	6.894	1.719
DS-C100	100	1-23	8.123	0

[1] Prozentualer Anteil von Beschreibungen aus duplikatfreien Quellen
[2] IDs der duplikatfreien Quellen
[3] Anz. Beschr. aus „sauberen“ Quellen
[4] Anz. Beschr. aus „schmutzigen“ Quellen

Tabelle 6.1.4: Aus DS-C erstellte Teilkorpora für die Evaluation von MSCD-ER

Quelle erfolgte dabei nicht zufällig, sondern mit Hilfe eines Similarity-Graphen. Die in Tabelle 6.1.2 abgebildete Blocking- und Matching-Konfiguration erzeugte einen Graphen für DS-C. Die Kanten des Graphen wurden mit der *Golden Truth* abgeglichen. Gehören beide Knoten, die durch eine Kante verbunden sind, in das gleiche Cluster der *Golden Truth*, so handelt es sich bei der Kante um ein korrektes Match (TP, engl. *True Positive*). Andernfalls ist das Match inkorrekt (FP, engl. *False Positve*). Nach Zählen von TP und FP lässt sich mit Hilfe der Formel $TP \cdot \frac{TP}{TP+FP}$ für jede Entitätsbeschreibung eine Kennzahl bestimmen, die das Verhältnis von korrekten und inkorrekten Kanten sowie die Anzahl der korrekten Kanten widerspiegelt. Die Beschreibung, für welche der Wert der Kennzahl in einem Cluster am größten ist, stellt den besten Vertreter seiner Quelle dar. Tabelle 6.1.3 zeigt die Anzahl der Entitätsbeschreibungen für jede Datenquelle in duplikatbehafteter sowie in dedublizierter Ausführung.

Mit Hilfe der dedublizierten Quellen wurden acht verschiedene Teilkorpora aus DS-C erzeugt, welche jeweils einen unterschiedlichen Anteil von Produktspezifikationen aus „sauberen“ Quellen enthalten, nach dem sie benannt sind. So enthält DS-C80 beispielsweise 80% Entitätsbeschreibungen aus sauberen Quellen. Tabelle 6.1.4 führt

die acht Teilkorpora mit den jeweils duplikatfrei enthaltenen Quellen auf. Die nicht in der dritten Spalte aufgelisteten Datenquellen liegen im jeweiligen Teilkorpus duplikatbehaftet vor. Die Wahl der Quellen erfolgte für die Korpora DS-C0, DS-C50, DS-C80 und DS-C100 so, dass der jeweils gewünschte Anteil von „sauberen“ Beschreibungen erreicht wird und ein breites Spektrum von 0% bis 100% duplikatfreien Teilkorpora entsteht. Mit DS-C26 und DS-C32 soll der besondere Einfluss einer sehr umfangreichen Quelle untersucht werden. Die größte Quelle in DS-C ist *ebay.com*. Sie enthält zwei Drittel der „schmutzigen“ und 40% der „sauberen“ Spezifikationen. Daher liegen in DS-C26 alle Quellen außer *ebay.com* duplikatfrei vor, während in DS-C32 ausschließlich *ebay.com* dedupliziert enthalten ist. Mit DS-C62A und DS-C62B lässt sich der Einfluss aller übrigen Quellen untersuchen. Die Korpora sind so zusammengestellt, dass ausgenommen von *ebay.com* alle Quellen, welche in DS-C62A duplikatfrei vorkommen, in DS-C62B duplikatbehaftet enthalten sind und umgekehrt. Dabei erreichen beide den gleichen Prozentanteil „sauberer“ Beschreibungen. Aus sämtlichen Similarity-Graphen aller Datensammlungen wurden Kanten zwischen zwei Entitätsbeschreibungen aus der gleichen duplikatfreien Quelle entfernt.

6.2 Evaluationsumgebung und Konfiguration

Die Experimente mit der umfangreichen Datensammlung DS-P fanden auf einem verteilten System mit der Shared-Nothing-Architektur statt. Es umfasst 18 Knoten, auf denen Apache Flink (vgl. Abschnitt 3.2) in der Version 1.9.0 und Apache Hadoop in der Version 2.6.0 installiert sind. Bei letzterem handelt es sich um ein Framework zur verteilten Datenverarbeitung, welches unter anderem das verteilte Dateisystem *Hadoop Distributed File System* (HDFS) zur Verfügung stellt. Hadoop wird in den Experimenten dieses Kapitels ausschließlich zur Speicherung der Daten genutzt und spielt bei der Verarbeitung keine Rolle. Die Datensammlungen liegen auf dem HDFS vor, werden durch Flink abgerufen, verarbeitet und im Ergebnis wieder auf dem HDFS gespeichert.

Bei 16 der Rechenknoten handelt es sich um Worker-Nodes, also Task-Manager in Flink und Data-Nodes in Hadoop. Die übrigen zwei Knoten übernehmen die Koordinationsaufgaben im verteilten System. Ein Knoten arbeitet als Job-Manager in Flink und als Name-Node in Hadoop. Der andere steht als Standby-System für beide Aufgaben bereit. Jeder Rechenknoten verfügt über einen Intel Xeon E5-2430 2,5 GHz Prozessor mit sechs Kernen, 48 Gigabyte Hauptspeicher und zwei SATA-Festplatten mit einer Kapazität von vier Terabyte. Als Betriebssystem kommt openSUSE 13.2

	DS-C0	DS-C26	DS-C32 DS-C50	DS-C62A DS-C62B	DS-C80	DS-C100
MSCD-AP Pref. CS	-	0,1	0,3	0,1	0,2	0,1
MSCD-AP Pref. DS	-	0,1	10^{-9}	0,1	0,1	-
AP Pref.	10^{-9}	0,1	0,1	0,1	0,1	0,1

Tabelle 6.2.1: Preference-Konfiguration für DS-C

zum Einsatz. Apache Flink wird mit sechs Task-Slots und 40 Gigabyte Hauptspeicher pro Worker-Node ausgeführt.

Für die weniger umfangreichen Datensammlungen DS-G, DS-M und DS-C fanden die Experimente auf einem einzelnen Desktop PC statt. Dieser ist mit einem AMD Ryzen 5 2600 3,4 GHz Prozessor mit sechs Kernen, 16 Gigabyte Hauptspeicher sowie einem 512 GB Solid-State-Drive ausgestattet. Der PC arbeitet mit dem Betriebssystem Kubuntu 19.10 und Apache Flink in der Version 1.7.2, auf welcher das FAMER-System derzeit noch aufbaut.

Für jeden der evaluierten Algorithmen wurde manuell die bestmögliche Parameter-Konfiguration ermittelt. In Abschnitt 5.1 sind die einzelnen Parameter der MSCD-AP Implementierungen aufgeführt und erläutert. HAP und MSCD-HAP verwenden einen fixen Preference-Wert von 0,7 in DS-G und 0,1 in DS-M und DS-P. Da bei den drei Datensammlungen sämtliche Datenquellen duplikatfrei vorliegen, ist keine Unterscheidung des Preference-Parameters für „schmutzige“ und „saubere“ Quellen erforderlich. In DS-C verwendet MSCD-AP meist einen größeren Preference-Wert für Datenpunkte aus duplikatfreien Quellen. Diese werden damit bevorzugt als Exemplar gewählt, wodurch bessere Ergebnisse erzielt werden können. Tabelle 6.2.1 zeigt die Preference-Konfiguration für DS-C.

Es mussten sehr geringe Werte für die Preference-Parameter gewählt werden, da AP im Allgemeinen dazu tendiert, zu viele Cluster zu erzeugen. Abbildung 6.2.1 zeigt dies anhand eines Beispiels. Frey und Dueck empfehlen, das Minimum der Ähnlich-

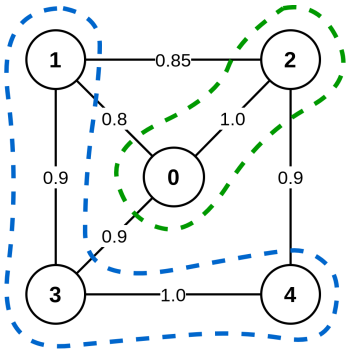


Abbildung 6.2.1: Zu starke Cluster-Separation in AP bei zu großem Preference-Wert

keitswerte als Preference-Parameter zu wählen, um eine geringe Cluster-Anzahl zu erhalten [12, S. 972]. Bei einer Preference von $0,8$ bildete AP jedoch die zwei auf der Abbildung farblich umrahmten Cluster. Erst bei einem Parameterwert von $\leq 0,76$ entsteht das erwartete Cluster mit allen fünf Beschreibungen. Die Ursache dafür ist, dass AP die Ähnlichkeitswerte nur anhand ihrer Unterschiede und nicht mittels des eigentlichen Wertes interpretieren kann. Sind wie im Beispiel nur Ähnlichkeitswerte in einem Bereich zwischen 80% und 100% vorhanden, dann betrachtet AP eine Ähnlichkeit von 80% als gering.

In allen Datensammlungen verwenden die AP-Derivate einen Dämpfungsfaktor von 0,5, einen *Damping-Adaption-Step* von 0,1, einen *Preference-Adaption-Step* von 0,05, 200 *Convergence-Iterations* sowie Rauschen ab der dritten Nachkommastelle. Für jede Datensammlung fanden Experimente mit verschiedenen Schwellwerten (SW) für die Datenpunktähnlichkeit statt, indem Kanten aus den Similarity-Graphen entfernt wurden, die den jeweiligen SW unterschritten. Der *All-Equal-Similarity-Threshold* ist für DS-G, DS-M und DS-P so eingestellt, dass er den geringsten Ähnlichkeitsschwellwert unterschreitet. In DS-C wird ein Wert von 0,49 verwendet. Für die kleinen Datensammlungen DS-G, DS-M und DS-C ist eine maximale Partitionsgröße von 1.000 eingestellt. Die Teilgraphen des Similarity-Graphen sind so klein, dass dadurch der hierarchische Ansatz nie zum Einsatz kommt und das Clustering ausschließlich mit den Basisalgorithmen AP und MSCD-AP erfolgt (ausgenommen ist DS-M mit SW 0,35). Die umfangreiche Datensammlung DS-P verfügt im Gegensatz dazu über viele sehr große Teilgraphen, wodurch ausgiebig vom hierarchischen Clustering-Ansatz Gebrauch gemacht wird. So verbinden in DS-P2 (SW 0,6) beispielsweise 1.430 Teilgraphen mehr als 400 Datenpunkte, wobei der größte Teilgraph 6.546 Knoten umfasst. Um die Experimente in einer akzeptablen Laufzeit ausführen zu können, wurde in DS-P eine maximale Partitionsgröße von 100 eingesetzt. Der folgende Abschnitt verwendet die Begriffe (H)AP und MSCD-(H)AP. Dabei handelt es sich um Überbegriffe, die sowohl die hierarchische als auch die Basisvariante des jeweiligen Algorithmus umfassen.

Die Parametereinstellungen der übrigen Algorithmen sind in Kapitel A beigelegt. Sämtliche Experimente wurden für die umfangreiche Datensammlung DS-P fünfmal und für DS-G, DS-M und DS-C zehnmal ausgeführt. In allen Diagrammen dieses Kapitels sind jeweils die Durchschnittswerte dieser Ausführungen dargestellt.

6.3 Einschränkungen

Die Evaluation muss sich auf die prototypische Implementierung von MSCD-HAP beschränken. Die beiden Prototypen für Sparse MSCD-AP sind zwar voll funktionsfähig, erfordern jedoch eine zu große Laufzeit und sind deshalb für den Einsatz im Big-Data-Umfeld nicht praktikabel. Das zeigt ein Experiment auf dem in Abschnitt 6.2 beschriebenen Rechencluster mit 16 Worker-Nodes und der kleinsten Datensammlung DS-G mit einem Schwellwert von 0,75. Tabelle 6.3.1 zeigt die Resultate bezüglich Laufzeit und Ergebnisqualtität der verschiedenen Prototypen. Eine Beschreibung der Maße Precision, Recall und F-Measure erfolgt im nächsten Abschnitt. Vorweggegriffen sei an dieser Stelle gesagt, dass die Ergebnisse der verschiedenen Prototypen, wie zu erwarten, nahezu identisch sind. Eine Laufzeit von fast drei bzw. mehr als zwölf Stunden für das Clustering von 3.054 Entitätsbeschreibungen ist jedoch inakzeptabel. Experimente mit den umfangreicheren Datensammlungen, welche 20.000 bis 10 Millionen Beschreibungen enthalten, sind daher für die Sparse MSCD-AP Implementierungen nicht sinnvoll und zeitlich auch nicht durchführbar. Alle weiteren Experimente dieses Kapitels beschränken sich deshalb auf die Evaluation des Prototyps für MSCD-HAP.

	MSCD-HAP	Sparse MSCD-AP	
		DataSet-API	Gelly
Laufzeit	30s	2h 42min	12h 30min
Precision	0,9967	0,9970	0,9974
Recall	0,9583	0,9699	0,9701
F-Measure	0,9771	0,9833	0,9836

Tabelle 6.3.1: Laufzeit und Ergebnisqualität der Prototypen für DS-G, SW 0,75

6.4 Qualität des Clusterings

Nachdem in den vorangegangenen Abschnitten die Evaluationsumgebung und die Datensammlungen vorgestellt wurden, beschäftigt sich der folgende Abschnitt mit der Effektivität des Clusterings. Dazu werden zunächst die verwendeten Metriken eingeführt. Anschließend erfolgt eine vergleichende Auswertung der Ergebnisse der sieben in Abschnitt 3.1.4 vorgestellten Clustering-Verfahren, die bereits in FAMER integriert sind (*Correlation Clustering CCPivot, CLIP, Center, Connected Components, Merge-Center, Star-1* und *Star-2 Clustering*), mit den neu implementierten Algorithmen HAP und MSCD-HAP. Als Erstes werden dabei die Resultate für die

drei MS-Clean-ER Datensammlungen ausgewertet und abschließend die Effektivität für MSCD-ER mit Hilfe der Teilkorpora von DS-C beurteilt.

Für die Ermittlung der Qualität von Clustering-Ergebnissen kommen häufig die Standard-Metriken Genauigkeit (engl. *Precision*), Trefferquote (engl. *Recall*) und das F-Maß (engl. *F-Measure*) zum Einsatz [30, 31, 38, 39]. Die genannten Metriken erfordern einen Abgleich der Match-Paare des Clustering Resultats mit jenen der Golden Truth. Ein Match-Paar enthält zwei Entitätsbeschreibungen, bei denen es sich mutmaßlich um die gleiche Entität der realen Welt handelt. Für jedes Cluster im Ergebnis wird jeweils ein Match-Paar für jede mögliche Kombination der Elemente des Clusters gebildet. Die Menge dieser Paare wird als „gefundene Matches“ bezeichnet. Gleiches erfolgt für die Cluster der *Golden Truth*, wobei die gebildeten Paare die Bezeichnung „tatsächliche Matches“ erhalten. Die „korrekt gefundenen Matches“ sind die Schnittmenge aus den gefundenen und den tatsächlichen Matches. In Formel 6.4.1 ergibt sich die Precision aus dem Anteil der korrekt gefundenen an allen ermittelten Matches. Recall ist in Formel 6.4.2 als der Anteil der korrekt gefundenen an den tatsächlichen Matches definiert. Das F-Measure bringt beide Metriken zusammen und bestimmt damit die Effektivität des Clusterings. Es ist möglich, Precision und Recall dabei unterschiedlich zu gewichten. In dieser Arbeit werden beide Metriken als gleich wichtig aufgefasst, wodurch sich das F-Measure entsprechend Formel 6.4.3 als das harmonische Mittel aus Precision und Recall ergibt. Das ungewichtete F-Measure wird auch als F_1 bezeichnet. [5, S. 167 f.]

$$\text{Precision} = \frac{\mid \text{gefundene Matches} \bigcap \text{tatsächliche Matches} \mid}{\mid \text{gefundene Matches} \mid} \tag{6.4.1}$$

$$\text{Recall} = \frac{\mid \text{gefundene Matches} \bigcap \text{tatsächliche Matches} \mid}{\mid \text{tatsächliche Matches} \mid} \tag{6.4.2}$$

$$\text{F-Measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.4.3}$$

Wie bereits im vorangegangenen Abschnitt erwähnt, fanden die Experimente mit verschiedenen Schwellwerten für die Datenpunktähnlichkeit statt. Dazu wurden die Kanten aus den Similarity-Graphen entfernt, deren Gewicht den jeweiligen Schwellwert unterschritt. Im Allgemeinen führt das Entfernen von Kanten mit geringen Ähnlichkeitswerten zu einer größeren Genauigkeit bei den meisten Clustering-Verfahren, da die verbleibenden Kanten nur noch die wichtigeren Ähnlichkeitsbeziehungen des Graphen darstellen. Es führt jedoch auch zu einem Rückgang der Trefferquote, weil entfernte Kanten für die Identifizierung korrekter Matches notwendig waren [38, S. 77 f.]. Als Vergleichsmaßstab ist in sämtlichen Diagrammen dieses Ab-

schnitts das Ergebnis des Similarity-Graphen aufgeführt, welches dieser ohne Clustering erzielt. Dabei wird jede vorhandene Kante als gefundenes Match betrachtet und transitive Relationen bleiben unbeachtet.

6.4.1 Multi-Source Clean Entity Resolution

Bevor der nächste Abschnitt die Ergebnisse von MSCD-ER in der Datensammlung DS-C analysiert, widmet sich dieser Abschnitt zunächst MS-Clean-ER mit Hilfe der Datensammlungen DS-G, DS-M und DS-P. Abbildung 6.4.1 zeigt die Ergebnisse der verschiedenen Clustering-Verfahren. Zugunsten einer besseren Übersichtlichkeit ist der Wertebereich der Y-Achsen in den Diagrammen eingeschränkt. In Kapitel A sind Balkendiagramme mit vollständigem Wertebereich beigelegt, welche ebenfalls die Standardabweichung der wiederholten Versuchsausführungen darstellen.

Das für den Anwendungsfall MS-Clean-ER spezialisierte CLIP-Verfahren erzielt in den meisten Fällen das beste Resultat im F-Measure. Abgesehen von der Musik-Domäne ist MSCD-(H)AP dem Spezialverfahren allerdings ebenbürtig und kann es in DS-P1 für geringe Schwellwerte sogar noch übertreffen. Das gute Resultat im F-Measure entstammt hauptsächlich den überragenden Ergebnissen bei der Precision. Hier ist MSCD-(H)AP gegenüber allen anderen Algorithmen deutlich überlegen. Dies gilt besonders für geringe Ähnlichkeitsschwellwerte, bei denen Connected Components, Merge-Center und Star-1 am schlechtesten abschneiden. Im Recall erzielen die meisten der anderen Algorithmen bessere Ergebnisse als MSCD-(H)AP. Das liegt daran, dass sie weniger und größere, aber ungenauere Cluster bilden. In diesen befinden sich zwar viele falsche, doch auch einige zusätzliche korrekte Beschreibungspaare (Matches), die von MSCD-(H)AP nicht erkannt werden konnten. Letzteres bildet hingegen eine größere Anzahl von kleineren, aber genaueren Clustern, um die Datenpunkte aus den „sauberen“ Quellen korrekt voneinander trennen zu können. Star-2, CCPivot und das traditionelle (H)AP erzielen mittelmäßige Ergebnisse bei der Precision, wobei (H)AP in DS-G und CCPivot in DS-M eine große Genauigkeit und damit auch ein gutes Resultat im F-Measure erzielen können.

Die herausragende Genauigkeit von MSCD-(H)AP lässt sich dadurch erklären, dass es neben CLIP als einziges von der Information über die duplikatfreien Quellen Gebrauch macht und somit Entitätsbeschreibungen der gleichen Quelle strikt voneinander trennt. Es kann CLIP im Hinblick auf die Precision übertreffen, da es sämtliche Ähnlichkeitswerte im Similarity-Graphen berücksichtigt. CLIP hingegen beachtet nur Kanten einer gewissen Stärke (vgl. Abschnitt 3.1.4). Die größte Genauigkeit unter den Clustering-Verfahren, welche das Clean-Source-Constraint außer Acht lassen, erzielt der Center-Algorithmus. Er erzeugt jedoch in DS-G und

DS-M zu viele kleine Cluster und erreicht dadurch nur eine geringe Trefferquote, die sich auf das Ergebnis im F-Measure niederschlägt.

Im Vergleich zum traditionellen (H)AP ist MSCD-(H)AP im Recall leicht unterlegen, aber in der Precision deutlich überlegen, wodurch es in allen Datensammlungen ein besseres Ergebnis im F-Measure erzielen kann. Der Unterschied bei der Genauigkeit ist besonders stark in der Personen-Domäne ausgeprägt. Dies erklärt

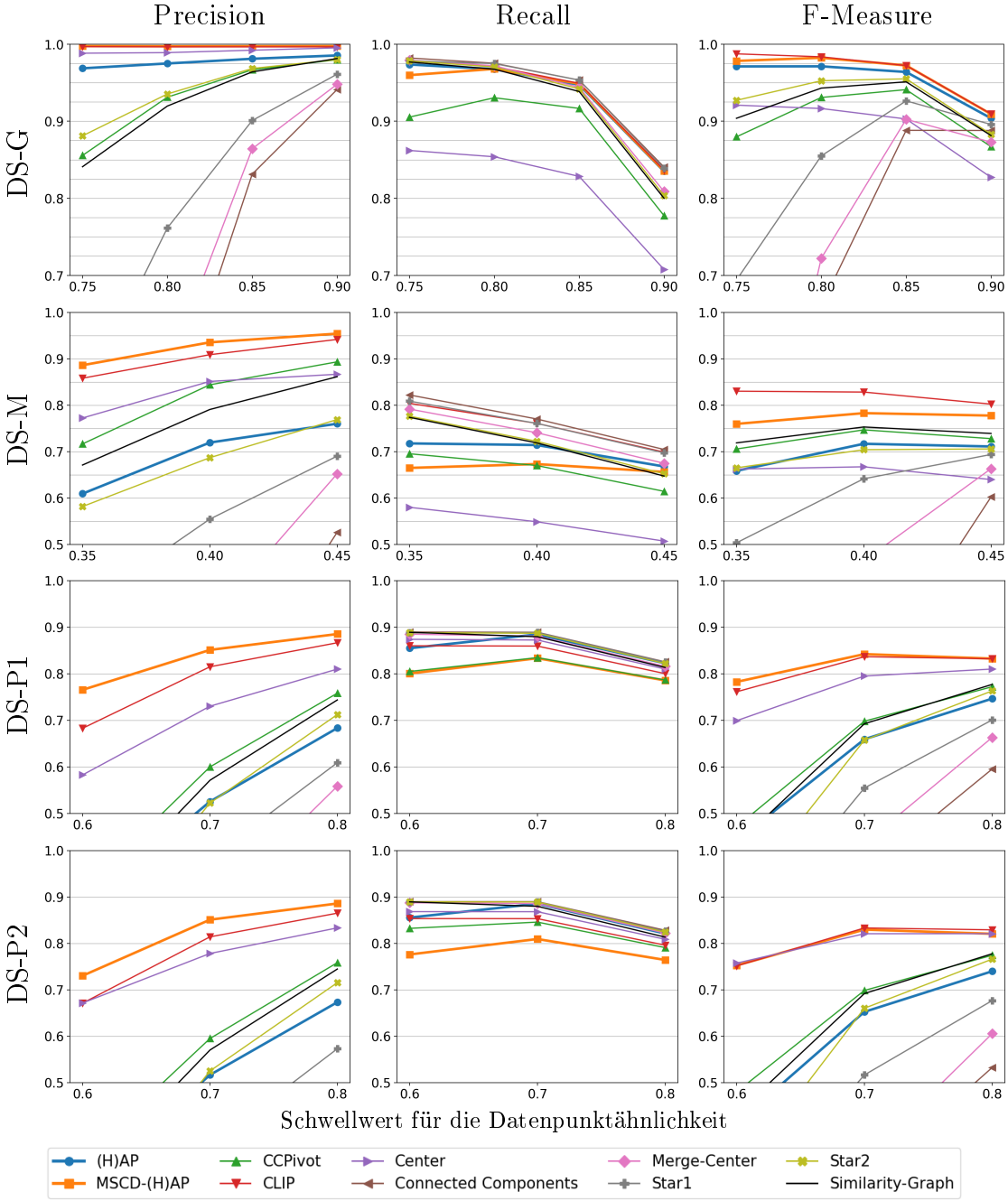
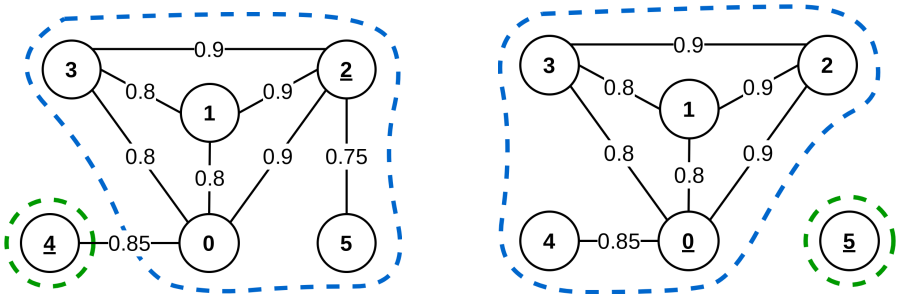


Abbildung 6.4.1: Clustering-Qualität für die MS-Clean-ER Datensammlungen



(a) Ähnlichkeitsschwellwert: 0,7 **(b)** Ähnlichkeitsschwellwert: 0,8

Abbildung 6.4.2: Clustering eines Similarity-Graphen in AP mit verschiedenen Ähnlichkeitsschwellwerten

sich dadurch, dass die umfangreichen Datensammlungen DS-P1 und DS-P2 über vergleichsweise wenige Quellen, aber sehr große Teilgraphen verfügen. Bei Nichtberücksichtigung der Clean-Source-Konsistenz entstehen große Cluster mit sehr vielen inkorrekten Matches, die aus Datenpunktpaaren der gleichen Quelle bestehen.

Wie bereits erwähnt, führt das Entfernen von Kanten durch einen größeren Ähnlichkeitsschwellwert bei den meisten Clustering-Algorithmen zu einem Rückgang der Trefferquote. Bei (H)AP und besonders bei MSCD-(H)AP ist im Gegansatz dazu oft sogar ein Anstieg zu beobachten. Dieses Verhalten lässt sich durch den Einfluss erklären, den die Ähnlichkeitswerte, welche bei einem größeren Schwellwert entfernt werden, auf die Wahl der Exemplare haben. Abbildung 6.4.2 zeigt dies anhand eines Beispiels. Mit einem Schwellwert von 0,7 besteht eine Kante zwischen Knoten zwei und fünf, die bei einem Schwellwert von 0,8 entfernt wird. Die Kante führt dazu, dass in Abb. 6.4.2a Knoten zwei zum Exemplar des blau markierten Clusters wird. Knoten vier hat keine Kante zu diesem Exemplar und muss daher ein eigenes Cluster bilden. In Abb. 6.4.2b versammelt Datenpunkt null aufgrund der entfernten Kante mehr Affinität als Knoten zwei auf sich und wird zum Exemplar des blauen Clusters. Dadurch kann Knoten vier ebenfalls diesem Cluster angehören. Handelt es sich bei den vier Knotenpaaren, die Knoten vier nun in dem Cluster bilden kann, um korrekte Matches, bedeutet dies einen Anstieg im Recall. Das Beispiel wurde mit einem Preference-Wert von 0,1 getestet.

6.4.2 Multi-Source Clean-Dirty Entity Resolution

Dieser Abschnitt widmet sich der Evaluation von MSCD-ER mit Hilfe der in Abschnitt 6.1 beschriebenen Teilkorpora von DS-C, welche jeweils einen unterschiedlichen Anteil von Entitätsbeschreibungen aus duplikatfreien Quellen enthalten. Dieser Anteil wird im Folgenden als *Clean-Portion* bezeichnet. Die Abbildungen 6.4.3 und 6.4.4 zeigen die Ergebnisse der verschiedenen Clustering-Verfahren für die Teilkorpora von DS-C. Wie auch für die im vorangegangenen Abschnitt ausgewerteten Datensammlungen sind die detaillierten Ergebnisse für DS-C in Kapitel A beigelegt.

Im Allgemeinen lässt sich für alle Teilkorpora sagen, dass sich MSCD-AP ähnlich wie bei den MS-Clean-ER Datensammlungen verhält. Es erzielt eine herausragende Genauigkeit, aber eine vergleichsweise geringe Trefferquote. Der Nachteil im Recall ist jedoch schwächer ausgeprägt als der Vorteil in der Precision, weshalb der Algorithmus ein vortreffliches Ergebnis im F-Measure erzielt. Ebenso wie AP, CLIP und Center erlangt MSCD-AP über verschiedene Ähnlichkeitsschwellwerte hinweg eine stabile Ergebnisqualität. Die übrigen Algorithmen benötigen hohe Schwellwerte für eine akzeptable Genauigkeit des Clusterings. Bei einem Schwellwert von 0,7 können CCPivot und die beiden Star-Clustering Varianten jedoch sehr gute Resultate im F-Measure erreichen und MSCD-AP teilweise sogar überbieten.

Bei Betrachtung der verschiedenen Clean-Portions zeigt sich, dass MSCD-AP für nahezu alle Teilkorpora überlegen ist. Die Überlegenheit ist besonders deutlich für das F-Measure bei geringen Ähnlichkeitsschwellwerten zu erkennen. Das einfache AP erzielt jedoch nur geringfügig schlechtere Ergebnisse. Das liegt hauptsächlich an den sehr genauen Ähnlichkeits- und Matching-Funktionen für DS-C (vgl. Tabelle 6.1.2) und der geringen Datenmenge. Durch die sehr exakten Ähnlichkeitswerte, die aus 12^3 verschiedenen Attributwerten gewonnen wurden, kann AP die einzelnen Entitätsbeschreibungen sehr gut unterscheiden. Im Vergleich zu DS-M und DS-P, bei denen die Genauigkeit von AP schlechter als die des Similarity-Graphen ausfällt (vgl. Abb. 6.4.1), übertrifft es diese für DS-C deutlich. In DS-G ist dies ebenso der Fall, was auf die hohe Genauigkeit der geographischen Distanz als Ähnlichkeitsmaß zurückzuführen ist. Des Weiteren erzeugt die Match-Funktion für DS-C relativ kleine Teilgraphen. So enthält beispielsweise der größte Teilgraph in DS-C50 mit einem Schwellwert von 0,3 genau 90 Entitätsbeschreibungen. Nur neun der Teilgraphen enthalten mehr als 50 Elemente. Wenn AP aus solchen Teilgraphen zu große Cluster bildet, wirkt sich das nicht so stark nachteilig auf die Precision aus wie beispielsweise bei der viel umfangreicheren Datensammlung DS-P. Es ist jedoch ein Trend erkennbar. Je geringer die Clean-Portion ist, umso ähnlicher sind sich die Ergebnis-

[3] X- und Y-Teil der Auflösung werden als ein Attribut betrachtet

se von AP und MSCD-AP. Wenn die Clean-Portion jedoch zunimmt, wachsen auch der Genauigkeitsvorteil und der Recall-Nachteil von MSCD-AP. Der Vorteil in der Genauigkeit wächst jedoch deutlicher als der Nachteil im Recall, weshalb die Überlegenheit im F-Measure umso stärker ausfällt, je mehr Beschreibungen aus sauberen Quellen entstammen. Für größere Datensammlungen mit umfangreicheren Teilgra-

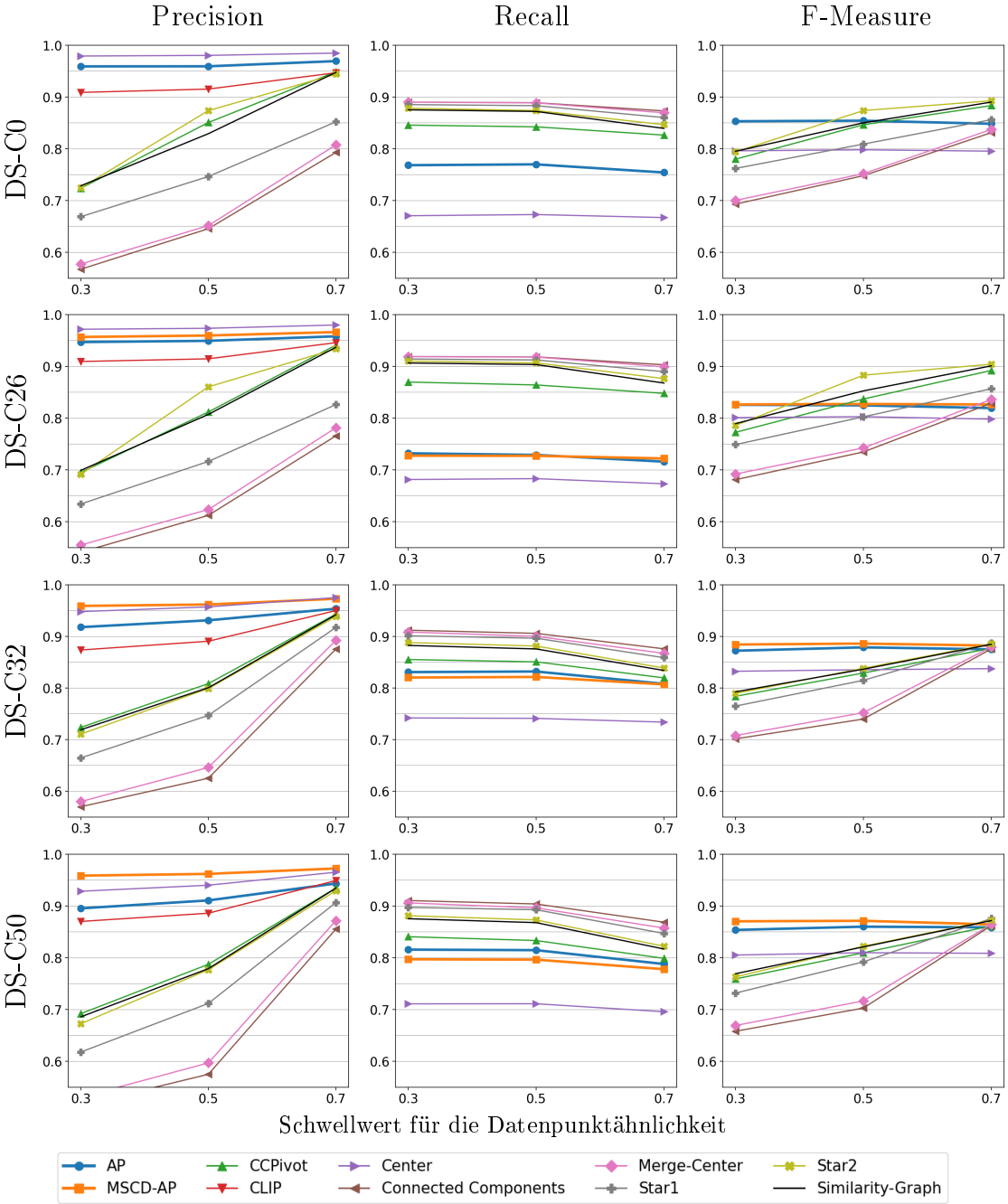


Abbildung 6.4.3: Clustering-Qualität von MSCD-ER für die Teilkorpora DS-C0 bis DS-C50

phen ist zu erwarten, dass dieser Vorteil, ähnlich wie in DS-P, deutlicher ausfallen wird.

In DS-C0, in dem keine duplikatfreien Quellen vorhanden sind, entspricht MSCD-AP dem klassischen AP. In DS-C100 sind ausschließlich „saubere“ Quellen enthalten, wodurch das für MS-Clean-ER spezialisierte *CLIP* das beste Ergebnis erzielen kann. Je geringer die Clean Portion ausfällt, desto schlechter schneidet *CLIP* ab, da der

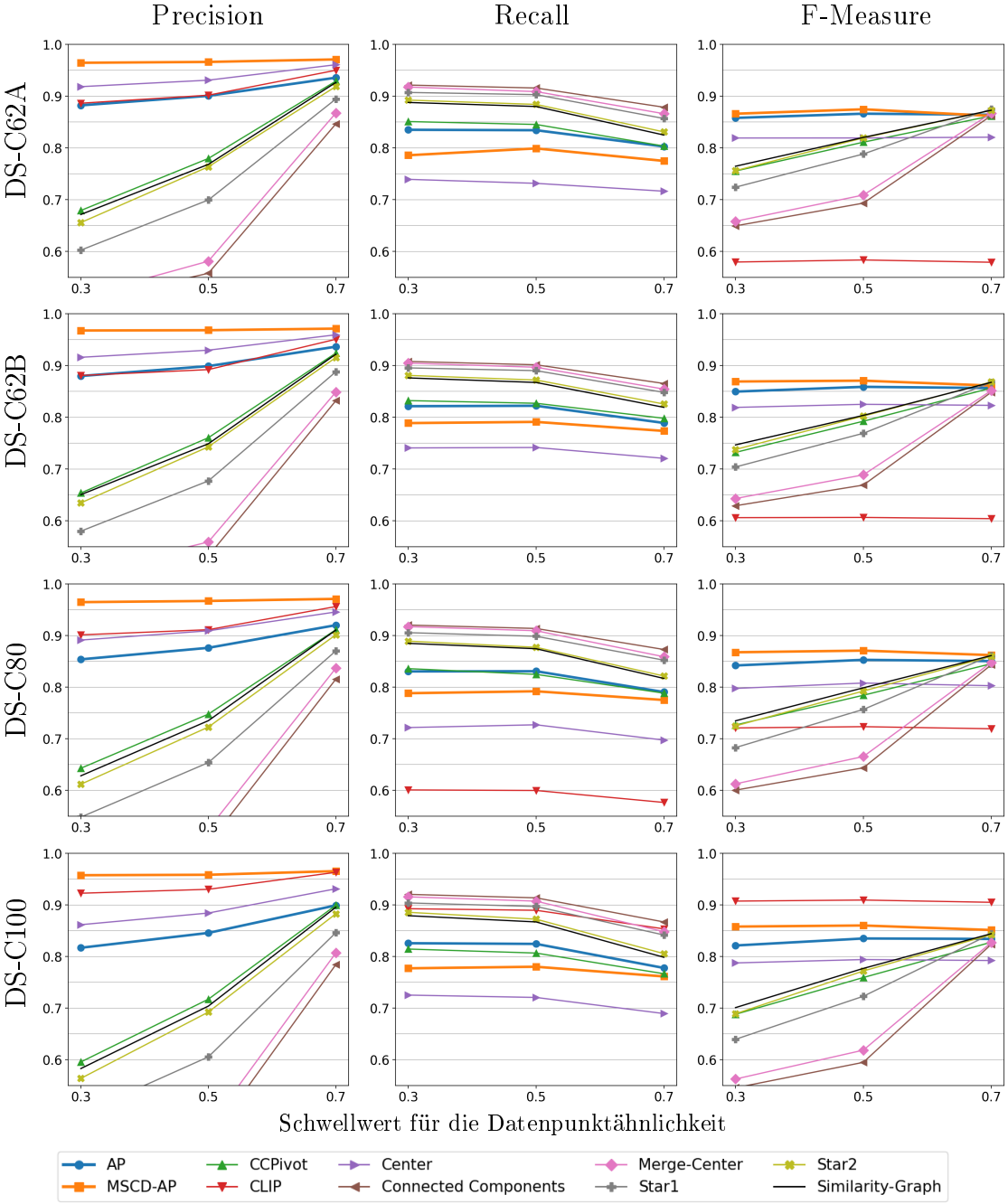


Abbildung 6.4.4: Clustering-Qualität von MSCD-ER für die Teilkorpora DS-C62A bis DS-C100

Algorithmus von der Annahme ausgeht, dass alle Quellen duplikatfrei vorliegen. Deshalb trennt er auch fälschlicherweise Datenpunkte aus duplikatbehafteten Quellen und erzielt damit einen schlechten Recall.

In DS-C26 und DS-C32 lässt sich der starke Einfluss der größten Quelle *ebay.com* erkennen. Wie in Abschnitt 6.1 erläutert, sind in DS-C26 alle Quellen dedupliziert, ausgenommen von *ebay.com*. In DS-C32 verhält es sich genau umgekehrt. Die Ergebnisse in DS-C26 ähneln denen in DS-C0 stark, da hier *ebay.com* 74% des Teilkorpus ausmacht. Das korrekte Trennen der Datenpunkte aus den übrigen Quellen kann die Precision des Ergebnisses nur geringfügig verbessern. In DS-C32, in dem die größte Quelle duplikatfrei vorliegt, zeigt sich, dass ein korrektes Trennen der Beschreibungen dieser Quelle deutlichere Vorteile in der Genauigkeit einbringen kann. Der Grund dafür ist, dass bei umfangreicheren Quellen mehr Datenpunkte der gleichen Quelle einem gemeinsamen Cluster zugeordnet sein können. Dies führt zu einer größeren Anzahl inkorrekter Matches.

DS-C62A und DS-C62B sollen den Einfluss der übrigen Quellen auf das Clustering-Ergebnis aufzeigen. Alle Quellen, die in DS-C62A duplikatfrei sind, liegen in DS-C62B duplikatbehaftet vor (ausgenommen von *ebay.com*) und umgekehrt. Die Clustering-Verfahren schneiden nahezu identisch in beiden Teilkorpora ab. Der Similarity-Graph in DS-C62A verfügt über eine geringfügig höhere Genauigkeit und Trefferquote, wovon die Multi-Source Dirty Clustering Algorithmen profitieren. In DS-C62B kann die Trennung von Beschreibungen aus duplikatfreien Quellen einen leicht größeren Vorteil bringen, wie sich am Ergebnis von MSCD-AP und CLIP erkennen lässt. Dennoch sind die übrigen Quellen ausgewogen, so dass sie sich gut für die Erzeugung von Teilkorpora mit verschiedenen Clean-Portions eignen und somit die Aussagekraft der Evaluation von MSCD-ER mit der Datensammlung DS-C bestätigen.

6.5 Laufzeitevaluation

Der folgende Abschnitt beschäftigt sich mit dem Laufzeitverhalten von MSCD-HAP und der Skalierbarkeit der Implementierung für die parallele Datenverarbeitung in einem Rechencluster. Da zur Beurteilung der Laufzeit große Datenmengen benötigt werden, findet die Evaluation ausschließlich auf der umfangreichen Datensammlung DS-P statt. Als Erstes werden die Laufzeiten der verschiedenen Clustering-Verfahren auf der Datensammlung verglichen. Anschließend erfolgt eine Analyse der Laufzeitbeschleunigung durch die parallele Ausführung von MSCD-HAP. Abschließend

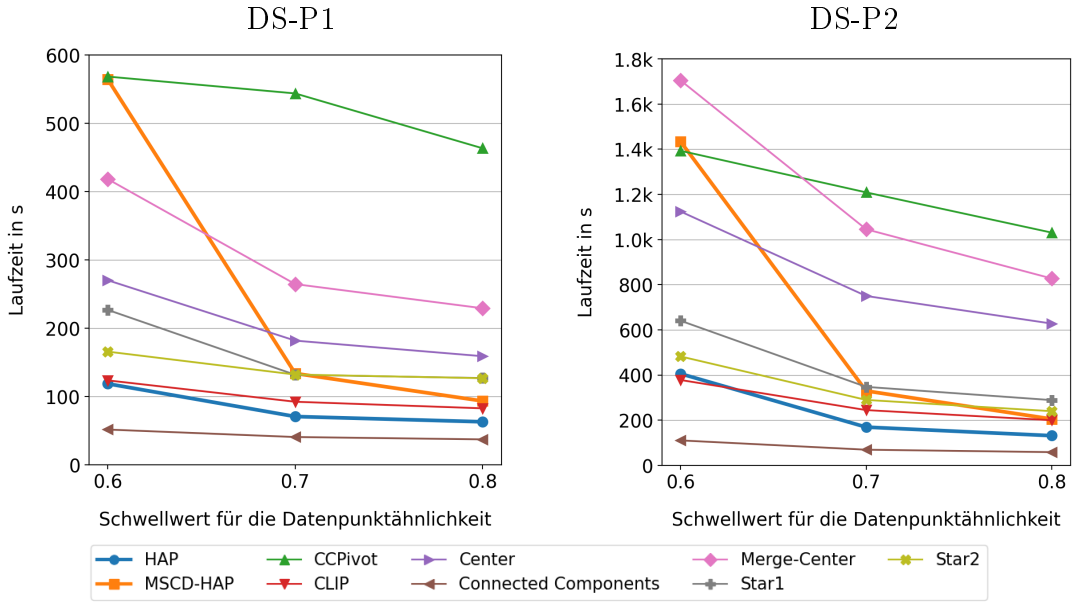


Abbildung 6.5.1: Laufzeiten der verschiedenen Clustering-Verfahren

werden Laufzeit und Ergebnisqualität bei verschiedenen Partitionsgrößen des hierarchischen Verfahrens untersucht.

Abbildung 6.5.1 stellt die durchschnittlichen Laufzeiten aus fünf Versuchsausführungen für die untersuchten Clustering-Verfahren dar. Dazu wurde das in Abschnitt 6.2 beschriebene Rechencluster mit 16 Worker-Nodes eingesetzt. Detaillierte Ergebnisse mit ausgewiesener Standardabweichung liegen in Kapitel B bei. Im Allgemeinen zeigt sich ein erwartetes Verhalten. Größere Datenmengen verursachen längere Rechenzeiten, so dass die Algorithmen mehr Zeit für den größeren Korpus DS-P2 benötigen. Höhere Ähnlichkeitsschwellwerte reduzieren die Laufzeiten, da die Similarity-Graphen dann weniger Kanten und kleinere Teilgraphen enthalten. Zur besseren Nachvollziehbarkeit dieses Umstandes sind in Tabelle 6.5.1 Informationen über Kanten und Teilgraphen für verschiedene Schwellwerte aufgeführt.

	DS-P1			DS-P2		
Schwellwert	0,6	0,7	0,8	0,6	0,7	0,8
#Knoten	5 Mio	5 Mio	5 Mio	10 Mio	10 Mio	10 Mio
#Kanten	10,3 Mio	5,1 Mio	3,6 Mio	46,7 Mio	23,1 Mio	16,4 Mio
#Teilgraphen (TG)	2,3 Mio	2,9 Mio	3,3 Mio	3,5 Mio	5,1 Mio	5,8 Mio
#TG $\geq$ 100 Knoten	2.841	141	28	8.350	764	196
#TG $\geq$ 200 Knoten	1.145	45	9	3.869	261	32
#TG $\geq$ 400 Knoten	328	13	4	1430	82	21
größter Teilgraph	3.301	1.072	990	6.546	2.625	2.204

Tabelle 6.5.1: Eigenschaften der Similarity-Graphen für verschiedene Ähnlichkeitsschwellwerte der Korpora von DS-P

MSCD-HAP benötigt ungefähr die zweieinhalbfache Zeit für die scheinbar doppelte Datenmenge (DS-P1 zu DS-P2, Schwellwerte 0,6 und 0,7). Verdoppelt ist in DS-P2 allerdings nur die Knotenanzahl. Die Menge der Kanten ist mehr als viermal größer als in DS-P1, wodurch auch der Umfang der Teilgraphen stark ansteigt. Verglichen mit den anderen Algorithmen ist der relative Anstieg des Zeitbefarfs von MSCD-HAP für das Clustering der größeren Datenmenge verhältnismäßig gering. Besonders die Algorithmen Center und Merge-Center, welche hauptsächlich mit den Kanten arbeiten, zeigen einen deutlichen Anstieg der Laufzeit. Sie benötigen ungefähr die vierfache Zeit für DS-P2 im Vergleich zu DS-P1. Connected Components ist das schnellste Verfahren und verzeichnet die geringste relative Laufzeitzunahme. Bei Schwellwert 0,6 benötigt der Algorithmus das 2,1-fache und bei größeren Schwellwerten weniger als das Doppelte der Rechenzeit.

MSCD-HAP ist deutlich langsamer als der Basisalgorithmus HAP. Die Gründe dafür sind vielfältig. Abschnitt 6.6 analysiert die Unterschiede beider Verfahren im Detail sowie deren Einfluss auf die Laufzeit und die Ergebnisqualität. Dem vorweggreifend ist für die vergleichende Betrachtung der Algorithmenlaufzeiten zu erwähnen, dass MSCD-HAP besonders bei der Verarbeitung umfangreicher Teilgraphen vergleichsweise langsame Rechenzeiten aufweist.

Für geringe Ähnlichkeitsschwellwerte gehört MSCD-HAP zusammen mit CCPivot und Merge-Center zu den langsamsten Clustering-Verfahren. Mit größeren Schwellwerten steigt jedoch die Geschwindigkeit von MSCD-HAP erheblich, so dass es zusammen mit Connected Components, AP, CLIP und den beiden Star-Clustering-Varianten zu den schnellsten Verfahren gehört. Dieser Unterschied liegt an der deutlich geringeren Anzahl großer Teilgraphen ab einem Ähnlichkeitsschwellwert von 0,7 (vgl. Tabelle 6.5.1). Da MSCD-HAP mit einer maximalen Partitionsgröße von 100 ausgeführt wurde, mussten bei einem Schwellwert von 0,6 ungefähr 20 Mal mehr umfangreiche Teilgraphen partitioniert werden als bei Schwellwert 0,7.

Speedup (engl. für Beschleunigung) ist ein Maß zur Beurteilung der „parallelen Performance“ [5, S. 218] eines Programms. Es gibt Auskunft darüber, wie sich die Laufzeit einer Anwendung durch die parallele Ausführung auf einem Multiprozessorsystem verbessern kann. Dazu wird die Rechenzeit zum Lösen eines Problems mit Hilfe von mehreren Prozessoren mit der Zeit verglichen, welche die Applikation mit einem einzelnen Prozessor benötigt. Zur Beurteilung der parallelen Performance eines Apache Flink Programms auf einem Rechencluster lässt sich die Anzahl der Worker-Nodes variieren. Der Speedup ergibt sich dann entsprechend Formel 6.5.1. Idealerweise wird bei p Worker-Nodes ein Speedup von p erreicht. [36, S. 177]

$$\text{Speedup}_p = \frac{\text{Laufzeit}(\#\text{Worker-Nodes} = 1)}{\text{Laufzeit}(\#\text{Worker-Nodes} = p)} \tag{6.5.1}$$

Abbildung 6.5.2 zeigt den Speedup von MSCD-HAP für die Ausführung mit einem, zwei, vier, acht und sechzehn Worker-Nodes auf den Korpora DS-P1 und DS-P2. Dabei ist zu berücksichtigen, dass Flink auf jedem der Worker-Nodes parallel in sechs Threads auf den einzelnen Prozessorkernen ausgeführt wird. Bei 16 Worker-Nodes arbeitet Flink also parallel mit 96 Threads. Dargestellt ist das Ergebnis der Laufzeitverbesserung für die verschiedenen Ähnlichkeitsschwellwerte, verglichen mit der idealen Beschleunigung. MSCD-HAP erreicht für beide Korpora von DS-P bis zu acht Worker-Nodes ein nahezu ideales Speedup-Ergebnis. Bei 16 Rechenknoten zeigt sich, dass für größere Datenmengen (DS-P2) und geringere Ähnlichkeitsschwellwerte (0,6) eine bessere Beschleunigung erzielt werden kann. Wie bereits oben beschrieben und in Tabelle 6.5.1 gezeigt, bedeutet ein geringerer Schwellwert eine größere Kantenanzahl sowie umfangreichere Teilgraphen und damit längere Laufzeiten. Der Rechenaufwand ist in DS-P1 mit höheren Schwellwerten nicht groß genug, um 16 Worker-Nodes voll ausnutzen zu können. Die Skalierbarkeit der übrigen Clustering-Verfahren in FAMER wird ausführlich von Saeedi et al. in [38] untersucht.

Die maximale Partitionsgröße ist der Parameter mit dem größten Einfluss auf die Laufzeit von MSCD-HAP. Abbildung 6.5.3 untersucht, wie sich sowohl Laufzeit als auch Ergebnisqualität bei einem Parameterwert von 100, 200 und 400 verhalten. Eine erhöhte Partitionsgröße führt zu einer besseren Genauigkeit, allerdings auch zu einer geringeren Trefferquote von MSCD-HAP. In größeren Partitionen kann MSCD-

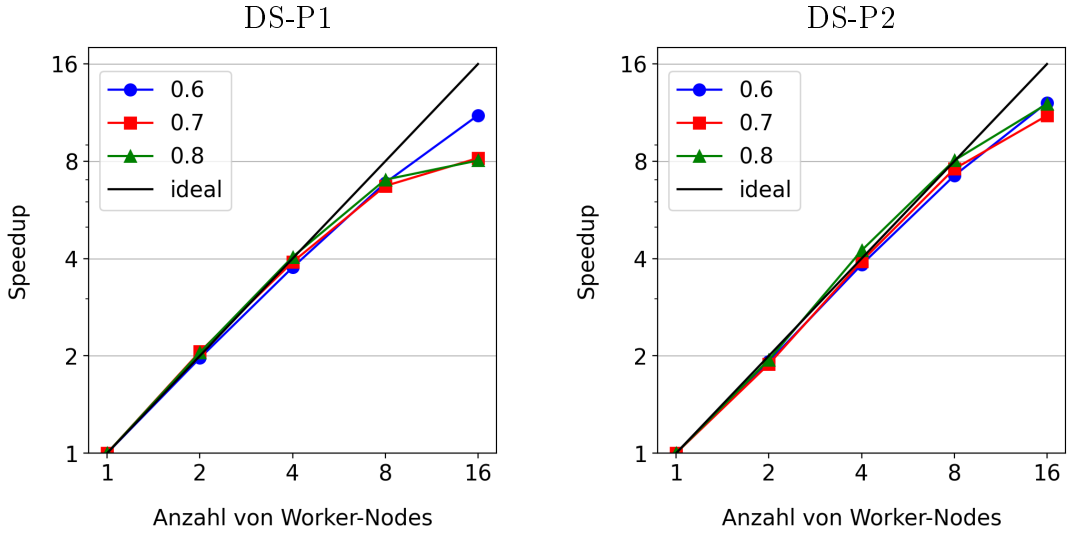


Abbildung 6.5.2: Speedup von MSCD-HAP für verschiedene Ähnlichkeitsschwellwerte (maximale Partitionsgröße: 100)

AP besser geeignete lokale und globale Exemplare identifizieren, wodurch am Ende genauere Cluster entstehen. Allerdings findet das Verfahren in größeren Partitionen auch mehr Exemplare, wodurch sich die Größe der resultierenden Cluster verringert und die Trefferquote sinkt. Das Erhöhen der Partitionsgröße verbessert die Precision mit sich abschwächendem Effekt, je weiter die Partitionen vergrößert werden. Umgekehrt verringert sich die Trefferquote mit zunehmendem Effekt bei steigender Partitionsgröße. Dies führt dazu, dass die Konfigurationen 200 und 400 ein nahezu identisches Ergebnis im F-Measure erzielen. Im Vergleich zur Partitionsgröße 100 konnte das Ergebnis jedoch verbessert werden. Unterschiede in der Ergebnisqualität sind fast ausschließlich für den geringsten Ähnlichkeitsschwellwert festzustellen, da bei diesem die Similarity-Graphen über umfangreichere Teilgraphen verfügen, die sich partitionieren lassen.

Die Laufzeit von MSCD-HAP erhöht sich mit zunehmender Partitionsgröße stark. Bei einem Schwellwert von 0,6 steigt sie um mehr als das Doppelte, wenn sich die Partitionsgröße verdoppelt. Für die Schwellwerte 0,7 und 0,8 zeigt sich ein deutlich stärkerer Laufzeitanstieg von Partitionsgröße 200 zu 400 als jener von 100 zu 200. Der Grund dafür liegt in der quadratischen Komplexität von MSCD-AP, welches zum Clustering der Partitionen eingesetzt wird. Die Laufzeit des Verfahrens wächst quadratisch mit der Partitionsgröße. Wenn das Clustering einer Partition aufgrund von Parameteradaptionen wiederholt werden muss, fällt dieser Umstand noch deutlicher ins Gewicht.

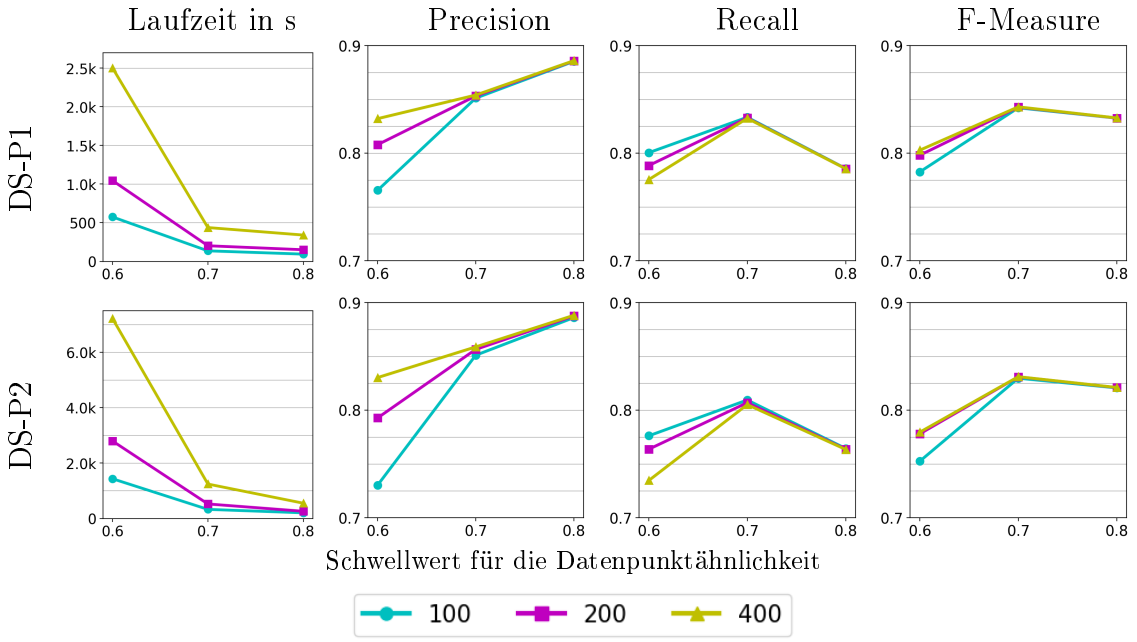


Abbildung 6.5.3: Laufzeit und Qualität des Clusterings von MSCD-HAP bei verschiedenen Einstellungen für die maximale Partitionsgröße

Obwohl MSCD-HAP mit den Partitionsgrößen 200 und 400 bessere Ergebnisse erzielen kann, ist in Abbildung 6.4.1 das Resultat unter Verwendung der Partitionsgröße 100 abgedruckt. Der Grund dafür ist, dass die vergleichende Evaluation der Laufzeit und der Ergebnisqualität für die verschiedenen Algorithmen mit der gleichen Konfiguration vorgenommen wurde, um einen fairen Vergleich der Verfahren zu ermöglichen.

6.6 Detailbetrachtung von MSCD-HAP und mögliche Varianten

Dieser Abschnitt untersucht den Einfluss der einzelnen Aspekte von MSCD-HAP auf die Laufzeit und die Ergebnisqualität mit Hilfe möglicher Varianten des Algorithmus. Diese Aspekte sind:

- die Exemplarzuordnung durch die Ungarischen Methode,
- die Verwendung von MSCD-AP für die Ermittlung von lokalen und globalen Exemplaren in den Partitionen,
- die Verwendung von MSCD-AP für das Clustering von Teilgraphen, welche die maximale Partitionsgröße unterschreiten.

Dazu wurden zwei Mischtypen aus HAP und MSCD-HAP erstellt. *Mix-Typ-1* verwendet AP anstelle von MSCD-AP sowohl für das Clustering der unpartitionierten als auch für die Exemplarermittlung auf den partitionierten Teilgraphen. Die Exemplarzuordnung für die globalen Exemplare der partitionierten Teilgraphen erfolgt jedoch mit Hilfe der Ungarischen Methode. *Mix-Typ-2* verwendet MSCD-AP für unpartitionierte und AP für partitionierte Teilgraphen. Auch in dieser Variante wird die Ungarische Methode eingesetzt. Abbildung 6.6.1 zeigt die Laufzeit und die Ergebnisqualität der MSCD-HAP-Varianten für das Clustering von DS-P bei einer maximalen Partitionsgröße von 400. Detaillierte Ergebnisse für die Partitionsgrößen 100, 200 und 400 mit ausgewiesener Standardabweichung liegen in Kapitel C bei.

Der einzige Unterschied zwischen HAP und Mix-Typ-1 ist die Verwendung der Ungarischen Methode bei der Exemplarzuordnung. Der Vergleich beider Varianten zeigt, dass die Ungarische Methode die Laufzeit kaum verlängert. Dafür ist der Einfluss auf die Genauigkeit des Clusterings enorm. Für DS-P2 und Schwellwert 0,6 verbessert sich die Precision in Mix-Typ-1 gegenüber HAP um 29%. Für geringere Partitionsgrößen ist der Unterschied sogar noch deutlicher (36% bei Partitionsgröße

100). Die Trefferquote verschlechtert sich beim Mix-Typ-1 nur geringfügig, so dass im F-Measure ebenfalls ein stark verbessertes Resultat erzielt werden kann.

Der Vergleich zwischen Mix-Typ-1 und Mix-Typ-2 zeigt auf, welchen Einfluss das Clustering der unpartitionierten Teilgraphen mit MSCD-AP auf die Laufzeit und die Ergebnisqualität nimmt. Bei großer Partitionsgröße und geringem Ähnlichkeitsschwellwert vervierfacht (DS-P2) bzw. verfünffacht (DS-P1) sich die Laufzeit durch den Einsatz von MSCD-AP. Bei geringeren Partitionsgrößen fällt der Unterschied wesentlich geringer aus. In DS-P2 mit Partitionsgröße 100 beträgt die Laufzeitzunahme nur 7%. MSCD-AP hat also deutlich größere Probleme beim Clustering umfangreicher Teilgraphen als der Basisalgorithmus AP. Die Trennung von Datenpunkten aus der gleichen, duplikatfreien Quelle führt zu einer größeren Exemplar- und Clusteranzahl. Die größere Exemplaranzahl führt zu vermehrten Oszillationen zwischen den verschiedenen Exemplaren. Dadurch benötigt das Verfahren mehr Parameteradaptionen, um eine konvergierende Parameterkonfiguration zu finden. Der Einsatz von MSCD-AP für die unpartitionierten Teilgraphen führt jedoch zu einer erheblichen Verbesserung der Precision. Bei einer Partitionsgröße von 400 steigt die Genauigkeit in DS-P2 um 22% und in DS-P1 sogar um 34%. Der Recall verschlechtert sich in einem geringeren Maße (9% DS-P1, 12% DS-P2), so dass Mix-Typ-2 ein deutlich verbessertes Resultat im F-Measure erzielen kann.

MSCD-HAP und Mix-Typ-2 unterscheiden sich dadurch, dass in letzterem AP anstelle von MSCD-AP auf den partitionierten Teilgraphen eingesetzt wird. Da MSCD-

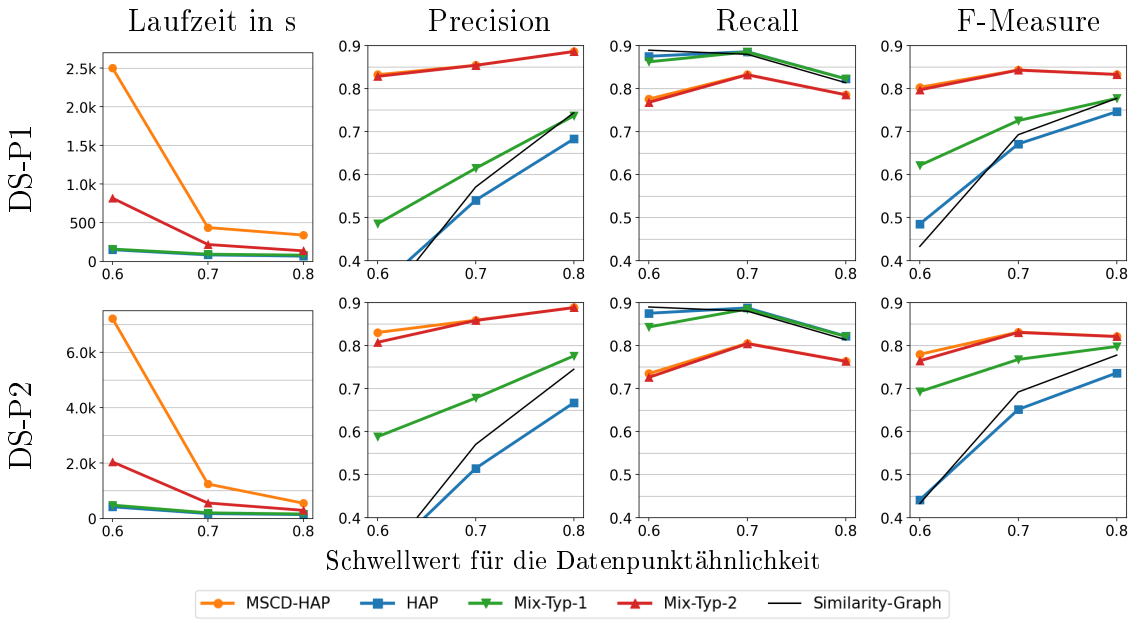


Abbildung 6.6.1: Laufzeit und Qualität des Clusterings für verschiedene Mischvarianten aus HAP und MSCD-HAP (maximale Partitionsgröße: 400)

AP, wie zuvor beschrieben, besonders in großen Teilgraphen durch die zusätzlichen Parameteradaptionen mehr Laufzeit als AP benötigt, verbraucht MSCD-HAP mehr als die dreifache Rechenzeit im Vergleich zu Mix-Typ-2. Hinsichtlich Genauigkeit und Trefferquote verbessert sich die Ergebnisqualität jedoch nur äußerst geringfügig. Das bedeutet, dass eine Berücksichtigung des Clean-Source-Constraints für die Wahl der lokalen und globalen Exemplare im hierarchischen Clustering kaum einen Vorteil einbringt. Wie in Abschnitt 4.3.1 beschrieben, kann MSCD-HAP die Datenquellen von zugeordneten Datenpunkten nicht über verschiedene Hierarchieebenen hinweg berücksichtigen. Da die finale Exemplarzuordnung durch die Ungarische Methode für eine strikte Einhaltung des Clean-Source-Constraints sorgt, kann AP zur Bestimmung der lokalen und globalen Exemplare eingesetzt werden, ohne das Ergebnis maßgeblich zu verschlechtern. Mix-Typ-2 ist also hinsichtlich der Effektivität des Clusterings ebenbürtig zu MSCD-HAP, kann es jedoch bezüglich der Effizienz deutlich übertreffen.

6.7 Zusammenfassung

In diesem Kapitel wurde eine ausgiebige Evaluation des neu entworfenen Clustering-Verfahrens für MSCD-AP durchgeführt, so dass die eingangs gestellten Fragen beantwortet werden konnten. Abschnitt 6.3 beantwortete die erste Frage und zeigte auf, dass nur einer der drei implementierten Prototypen für den Einsatz im Big-Data-Umfeld geeignet ist. Trotz einer hervorragenden Ergebnisqualität der beiden Implementierungen von Sparse MSCD-AP, können die Apache Flink Programme aufgrund deutlich zu schlechter Laufzeiten nicht für große Datenmengen eingesetzt werden.

MSCD-HAP stellte sich hingegen als ein hervorragender, universell für verschiedene Multi-Source-ER-Problemstellungen einsetzbarer Clustering-Algorithmus heraus. Das Verfahren zeigt sich besonders vorteilhaft beim Einsatz in MSCD-ER. In den Experimenten des Abschnitts 6.4.2 konnte MSCD-HAP für fast alle Clean-Portions das beste Resultat unter den verglichenen Algorithmen im F-Measure erzielen. Je höher der Anteil von Beschreibungen aus „sauberen“ Quellen war, umso größer zeigte sich der Vorteil von MSCD-HAP.

Aufgrund der strikten Trennung von Datenpunkten, welche aus der gleichen duplikatfreien Quelle stammen, ist MSCD-HAP auch exzellent als Clustering-Algorithmus für MS-Clean-ER geeignet. Die Experimente in Abschnitt 6.4.1 zeigten, dass nur das für diesen Anwendungsfall spezialisierte *CLIP*-Verfahren bessere Ergebnis-

se erzielen kann. MSCD-HAP erzielt die zweitbesten Resultate und kann *CLIP* in seltenen Fällen sogar übertreffen.

Im Vergleich mit dem Basisalgorithmus HAP zeichnet sich MSCD-HAP, aufgrund des neuen Clean-Source-Constraints und der Verwendung der Ungarischen Methode, durch eine stark verbesserte Genauigkeit aus. Die Präzision des Clusterings übertrifft sogar die sämtlicher anderer Algorithmen in FAMER. Abgesehen von DS-C26 erzielt es in jedem Experiment die beste Precision und kann besonders bei der umfangreichen Datensammlung DS-P mit Abstand die genausten Cluster bilden. Im Vergleich zu HAP bildet MSCD-HAP eine größere Anzahl von Clustern, die jeweils weniger Elemente enthalten. Das wirkt sich negativ auf den Recall aus. Hinsichtlich der Trefferquote zählt MSCD-HAP zu den schwächsten Verfahren. Dennoch ist der Nachteil im Recall deutlich geringer ausgeprägt als der Vorteil bei der Precision, so dass es HAP sowie die meisten übrigen Verfahren im F-Measure übertrifft.

Die Stärke von MSCD-HAP liegt besonders bei Similarity-Graphen mit geringen Ähnlichkeitsschwellwerten. Die meisten der anderen Algorithmen erfordern einen hohen Schwellwert, um eine gute Clustering-Qualität zu erreichen und MSCD-HAP teilweise zu übertreffen. Letzteres erzielt jedoch bei sämtlichen getesteten Schwellwerten eine hervorragende Ergebnisqualität. Doch die Qualität erlangt das Verfahren auf Kosten der Laufzeit. Bei geringen SW zählt es zu den langsamsten, bei größeren SW hingegen zu den schnellsten Verfahren. Abschnitt 6.5 zeigt, dass der MSCD-HAP Prototyp gut für große Datenmengen skaliert. Bei der parallelen Ausführung in einem Rechencluster mit mehreren Worker-Nodes erzielt das Flink-Programm einen nahezu idealen Speedup, wenn die gestellte Aufgabe umfangreich genug ist.

Die Partitionsgröße von MSCD-HAP hat einen starken Einfluss auf die Laufzeit und die Ergebnisqualität. Mit zunehmender Partitionsgröße kann eine höhere Genauigkeit, aber eine geringere Trefferquote erzielt werden. Der Recall fällt in der umgekehrten Intensität ab, in der die Precision ansteigt. Daher ist eine moderate Partitionsgröße für die beste Ergebnisqualität erforderlich. Die Laufzeit des Verfahrens wächst exponentiell mit der Partitionsgröße.

Abschnitt 6.6 untersuchte den Einfluss der einzelnen Aspekte von MSCD-HAP auf die Laufzeit und die Ergebnisqualität mit Hilfe verschiedener Mischformen aus HAP und MSCD-HAP. Eine Mischform, welche AP anstelle von MSCD-AP für die Bestimmung der lokalen und globalen Exemplare beim hierarchischen Clustering einsetzte, erreichte hinsichtlich der Effektivität nahezu das Niveau von MSCD-HAP. Die Mischform kann jedoch eine deutlich bessere Laufzeit erzielen und stellt daher eine effizientere Alternative zu MSCD-HAP dar.

Kapitel 7

Zusammenfassung und Ausblick

Obwohl Entity Resolution seit mehr als dreißig Jahren ein Thema wissenschaftlicher Untersuchungen ist, bleibt es weiterhin ein aktives Forschungsgebiet [7, S. 34]. In der vorliegenden Masterarbeit wurde ein bislang von der Forschung noch nicht berücksichtigter Zweig von ER aufgezeigt und mit dem Namen Multi-Source Clean-Dirty ER versehen. Ziel von MSCD-ER ist es, Duplikate in einer Datensammlung zu identifizieren, welche aus einer Vielzahl verschiedener Quellen besteht. Dabei ist von einigen dieser Quellen bekannt, dass diese frei von Intra-Source-Duplikaten sind, also in sich duplikatfrei vorliegen. Für MSCD-ER existierte bisher noch kein spezialisiertes Clustering-Verfahren, welches das Auflösen der Entitäten in einem einzigen Durchlauf des ER-Prozesses ermöglichte. Diese Lücke wurde mit der vorliegenden Masterarbeit geschlossen. Mit MSCD-AP konnte ein spezialisierter Clustering-Algorithmus entwickelt werden, der die Quellqualität hinsichtlich der Duplikatfreiheit berücksichtigt und Entitätsbeschreibungen der gleichen „sauberen“ Quelle strikt und zuverlässig trennt.

In einer Problemanalyse wurde zunächst ein Schema von Christophides et al. untersucht, welches ER in verschiedene Typen unterteilt, denen wiederum spezialisierte Clustering-Verfahren zugeordnet sind. Dieses Schema beruht auf zwei Charakteristika der aufzulösenden Datensammlung: der Anzahl und der Duplikatfreiheit der Datenquellen. Das Schema ließ sich im Rahmen der Problemanalyse weiter aufgliedern, so dass Multi-Source-ER in MS-Clean-ER, MS-Dirty-ER und MSCD-ER unterteilt wurde. Bei Zuordnung der bestehenden Clustering-Algorithmen zu den drei neuen Zweigen fiel auf, dass für MSCD-ER kein spezialisiertes Verfahren existierte. Anschließend wurden die Anforderungen an einen Clustering-Algorithmus spezifiziert, die dieser erfüllen muss, um der neuen Kategorie MSCD-ER zuordenbar zu sein.

Die nachfolgende Analyse verwandter Arbeiten schuf zunächst ein grundlegendes Verständnis von ER im Big-Data-Umfeld und stellte die beiden Systeme Apache Flink und FAMER vor, deren Kenntnis für die Implementierung eines Prototypen

von MSCD-AP bedeutsam war. Eine detaillierte Auseinandersetzung mit dem Basisalgorithmus Affinity Propagation sowie der ihm zugrunde liegenden Konzepte, dem Max-Sum-Algorithmus und Faktorgraphen, ermöglichte die Nutzung der Erweiterbarkeit des Verfahrens und schließlich die Entwicklung von MSCD-AP.

Ausgehend von den vorgestellten Konzepten konnte mit dem Clean-Source-Constraint eine Erweiterung für den Faktorgraphen von AP entworfen werden, welche den Algorithmus um die Funktionalität bereichert, Datenpunkte aus beliebigen duplikatfreien Quellen konsequent voneinander zu separieren. Aus dem Max-Sum-Algorithmus ließen sich Formeln für die auf dem Faktorgraphen ausgetauschten Nachrichten ableiten. Somit approximiert der neu entworfene Algorithmus MSCD-AP das Optimierungsproblem einer optimalen Auswahl von Cluster-Zentren und einer Zuordnung der restlichen Datenpunkte zu diesen Exemplaren, unter Wahrung der Clean-Source-Konsistenz. Da der Basisalgorithmus AP und damit auch MSCD-AP eine quadratische Komplexität aufweisen, wurden im Anschluss die zwei Ansätze Sparse MSCD-AP und MSCD-HAP entwickelt, um ein skalierbares Clustering für große Datenmengen zu ermöglichen.

Basierend auf Apache Flink wurden drei Prototypen entwickelt, zwei für Sparse MSCD-AP und einer für MSCD-HAP. Eine Integration der Prototypen als Clustering-Module in das ER-System FAMER erlaubt die Ausführung eines vollständigen MSCD-ER Prozesses und eine vergleichende Evaluation mit den anderen Clustering-Algorithmen des Systems.

Die Evaluation erfolgte anhand dreier bereits vorhandener Datensammlungen für MS-Clean-ER und einer neu entworfenen Sammlung aus mehreren Teilkorpora mit unterschiedlichen Clean-Portions, welche auch zukünftig als Benchmark für MSCD-ER dienen kann. Es zeigte sich, dass die beiden Prototypen für Sparse MSCD-AP in ihrer aktuellen Form nicht für das Clustering großer Datenmengen geeignet sind. Die Flink-Implementierungen haben zu große Defizite bei der Laufzeit. Der Prototyp von MSCD-HAP hingegen zeigte ein akzeptables Laufzeitverhalten beim Clustering umfangreicher und ein exzellentes bei weniger umfangreichen Teilgraphen, welche bei größeren Schwellwerten für die Datenpunktähnlichkeit entstehen. Zudem erzielte er eine hervorragende Clustering-Qualität bei allen getesteten Datensammlungen. Verglichen mit den in FAMER vorhandenen Algorithmen war MSCD-HAP der einzige, welcher bei jeder Datensammlung, jeder Clean-Portion und jedem Ähnlichkeitsschwellwert sehr gute Clustering-Ergebnisse erzielen konnte. Er eignet sich daher als universelles Clustering-Werkzeug für die verschiedensten Aufgabenstellungen von Entity Resolution. Besonders hervorstechend war die Genauigkeit des Verfahrens. MSCD-HAP erzielte fast immer, und häufig sogar mit Abstand, die ge-

nauesten Clustering-Resultate. Einzig der Center-Algorithmus konnte die Precision des neuen Verfahrens bei sehr geringen Clean-Portions übertreffen. Bei der Trefferquote gehörte MSCD-HAP zu den schlechtesten der evaluierten Algorithmen. Die Schwäche war jedoch weniger stark ausgeprägt als die Stärke in der Precision, wodurch sich eine hervorragende Clustering-Qualität beim F-Measure ergab. Bei der parallelen Ausführung auf einem Rechencluster zeigte MSCD-HAP ein nahezu optimales Ergebnis im Speedup, solange die Problemstellung umfangreich genug war. Die maximale Partitionsgröße ergab sich als wichtigster Parameter für die Laufzeit und die Ergebnisqualität.

Obwohl MSCD-HAP sämtliche Anforderungen an ein spezialisiertes Clustering-Verfahren für MSCD-ER erfüllt, hat sich aber auch gezeigt, dass der Algorithmus noch stark optimiert werden kann. Ein erster Ansatz dafür war ein Mischtyp aus HAP und MSCD-HAP, der den Basisalgorithmus Affinity Propagation für die Ermittlung der lokalen und globalen Exemplare in den partitionierten Teilgraphen einsetzte. Der Mischtyp konnte eine nahezu ebenbürtige Clustering-Qualität erzielen, gleichzeitig aber die Laufzeit des Algorithmus deutlich verbessern. Die vollständige Implementierung von Adaptive AP [50] ist vielversprechend, um zukünftig sowohl die Laufzeit als auch die Ergebnisqualität von MSCD-HAP weiter zu verbessern. Die Parameteradaption „von innen“, während der Ausführung des Algorithmus, verspricht große Zeiteinsparungen, verglichen mit der derzeit angewendeten Adaption „von außen“. Bei letzterer muss das Clustering eines Teilgraphen mit angepassten Parametern neu gestartet werden, falls es für keine valide Lösung konvergiert ist. Adaptive AP hingegen erkennt Oszillationen und passt die Parameter während der Ausführung des iterativen Message-Passings an. Zusätzlich verspricht Adaptive AP durch den Einsatz des sogenannten *p-scannings* eine bessere Clustering-Qualität. Falls das Verfahren eine Lösung gefunden hat, bricht es das Message-Passing nicht ab, sondern passt die Parameter an und sucht nach einer möglicherweise noch besseren Lösung. Ein weiterer Ansatzpunkt für die Verbesserung der Ergebnisqualität ist die Partitionierung in MSCD-HAP. Anstatt diese wie bisher zufällig vorzunehmen, könnte sie aufgrund gewisser Kriterien erfolgen. So wäre es beispielsweise denkbar, ähnlich wie CLIP, Kantencharakteristika auszuwerten, um dafür zu sorgen, dass die Knoten einer Partition möglichst viele gemeinsame Kanten aufweisen.

MSCD-HAP muss nicht der einzige Clustering-Algorithmus für MSCD-ER bleiben. Zukünftige Forschung könnte versuchen, andere Algorithmen um neue Funktionalitäten zu erweitern, so dass sie die Duplikatfreiheit der Quellen beim Clustering berücksichtigen und die Clean-Source-Konsistenz wahren. Als Benchmark für künftige MSCD-Clustering-Verfahren kann die für diese Arbeit erstellte Datensammlung

aus der Digitalkamera-Domäne dienen. Die schlechte Laufzeit der beiden Prototypen von Sparse MSCD-AP bedeuten nicht, dass der Ansatz völlig verworfen werden muss. Möglicherweise lässt sich in künftigen Arbeiten ein besserer Implementierungsansatz finden, der Sparse MSCD-AP performant in Flink umsetzen kann.

Alles in Allem ist mit MSCD-HAP der Entwurf und die Implementierung eines spezialisierten Clustering-Algorithmus für MSCD-ER gelungen. Das Verfahren weist eine hervorragende Effektivität sowie eine hinreichende Effizienz auf und ist universell für verschiedene Typen von ER einsetzbar. Die Clustering-Qualität und die Laufzeit des Algorithmus lassen sich voraussichtlich weiter verbessern, wenn die in diesem Abschnitt vorgeschlagenen Optimierungsansätze verfolgt werden.

Literatur

[1] Aslam, Javed A.; Pelekhov, Ekaterina und Rus, Daniela: The star clustering algorithm for static and dynamic information organization. In: *Journal of Graph Algorithms and Applications* Vol. 8.1, 2004, S. 95–129. DOI: 10.7155/jgaa.00084.

[2] Bansal, Nikhil; Blum, Avrim und Chawla, Shuchi: Correlation clustering. In: *Machine learning* Vol. 56.1-3, Springer, 2004, S. 89–113. DOI: 10.1023/B:MACH.0000033116.57574.95.

[3] Carbone, Paris; Katsifodimos, Asterios; Ewen, Stephan; Markl, Volker; Haridi, Seif und Tzoumas, Kostas: Apache flink: Stream and batch processing in a single engine. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* Vol. 36.4, 2015.

[4] Chierichetti, Flavio; Dalvi, Nilesh und Kumar, Ravi: Correlation clustering in mapreduce. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, S. 641–650. DOI: 10.1145/2623330.2623743.

[5] Christen, Peter: Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection. 1. Auflage. Berlin Heidelberg: Springer-Verlag, 2012. ISBN: 978-3-642-31164-2.

[6] Christen, Peter und Vatsalan, Dinusha: Flexible and extensible generation and corruption of personal data. In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, S. 1165–1168. DOI: 10.1145/2505515.2507815.

[7] Christophides, Vassilis; Efthymiou, Vasilis; Palpanas, Themis; Papadakis, George und Stefanidis, Kostas: End-to-End Entity Resolution for Big Data: A Survey. In: *arXiv preprint arXiv:1905.06397*, 2019.

[8] Cramer, Erhard und Kamps, Udo: Grundlagen der Wahrscheinlichkeitsrechnung und Statistik. Eine Einführung für Studierende der Informatik, der Ingenieur- und Wirtschaftswissenschaften. 5. Auflage. Berlin: Springer Spektrum, 2020. ISBN: 978-3-662-60552-3.

[9] Cui, Hong; Zhang, Jingjing; Cui, Chunfeng und Chen, Qinyu: Solving large-scale assignment problems by Kuhn-Munkres algorithm. In: *2nd International Conference on Advances in Mechanical Engineering and Industrial Informatics (AMEII)*, Hangzhou, Zhejiang, 2016, S. 822–827. DOI: 10.2991/ameii-16.2016.160.

[10] Dean, Jeffrey und Ghemawat, Sanjay: MapReduce: simplified data processing on large clusters. In: *Communications of the ACM* Vol. 51.1, ACM New York, NY, USA, 2008, S. 107–113. DOI: 10.1145/1327452.1327492.

[11] Dueck, Delbert: Affinity Propagation: Clustering Data by Passing Messages. Doktorarbeit. University of Toronto, 2009.

[12] Frey, Brendan J. und Dueck, Delbert: Clustering by passing messages between data points. In: *science* Vol. 315.5814, American Association for the Advancement of Science, 2007, S. 972–976. DOI: 10.1126/science.1136800.

[13] Frey, Brendan J. und Dueck, Delbert: Supporting online material for clustering by passing messages between data points. In: *science* Vol. 315.5814, American Association for the Advancement of Science, 2007. https://science.sciencemag.org/content/suppl/2007/01/08/1136800.DC1.

[14] Frey, Brendan J.; Kschischang, Frank R.; Loeliger, Hans-Andrea und Wiberg, Niclas: Factor graphs and algorithms. In: *Proceedings of the Annual Allerton Conference on Communication Control and Computing* Vol. 35, University of Illinois, 1997, S. 666–680.

[15] Furtlehner, Cyril; Sebag, Michèle und Zhang, Xiangliang: Scaling analysis of affinity propagation. In: *Physical review E* Vol. 81.6, APS, 2010, S. 066102. DOI: 10.1103/PhysRevE.81.066102.

[16] Givoni, Inmar-Ella: Beyond affinity propagation: Message passing algorithms for clustering. Doktorarbeit. University of Toronto, 2012.

[17] Givoni, Inmar-Ella; Chung, Clement und Frey, Brendan J.: Hierarchical affinity propagation. In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, 2011, S. 238–246.

[18] Givoni, Inmar-Ella und Frey, Brendan J.: A binary variable model for affinity propagation. In: *Neural computation* Vol. 21.6, MIT Press, 2009, S. 1589–1600. DOI: 10.1162/neco.2009.05-08-785.

[19] Hassanzadeh, Oktie und Miller, Renée J.: Creating probabilistic databases from duplicated data. In: *The VLDB Journal* Vol. 18.5, Springer, 2009, S. 1141–1166. DOI: 10.1007/s00778-009-0161-2.

[20] Hernández, Mauricio A. und Stolfo, Salvatore J.: Real-world data is dirty: Data cleansing and the merge/purge problem. In: *Data mining and knowledge discovery* Vol. 2.1, Springer, 1998, S. 9–37. DOI: 10.1023/A:1009761603038.

[21] Hernández, Mauricio A. und Stolfo, Salvatore J.: The merge/purge problem for large databases. In: *ACM Sigmod Record* Vol. 24.2, ACM New York, NY, USA, 1995, S. 127–138. DOI: 10.1145/568271.223807.

[22] Hildebrandt, Kai; Panse, Fabian; Wilcke, Niklas und Ritter, Norbert: Large-Scale Data Pollution with Apache Spark. In: *IEEE Transactions on Big Data* Vol. 6.2, 2020, S. 396–411. DOI: 10.1109/TBDATA.2016.2637378.

[23] Khan, Rayyan Ahmad; Amjad, Rana Ali und Kleinsteuber, Martin: Extended Affinity Propagation: Global Discovery and Local Insights. In: *arXiv preprint arXiv:1803.04459*, 2019.

[24] Kolb, Lars; Thor, Andreas und Rahm, Erhard: Load balancing for mapreduce-based entity resolution. In: *2012 IEEE 28th international conference on data engineering*, IEEE, 2012, S. 618–629. DOI: 10.1109/ICDE.2012.22.

[25] Kschischang, Frank R.; Frey, Brendan J. und Loeliger, Hans-Andrea: Factor graphs and the sum-product algorithm. In: *IEEE Transactions on information theory* Vol. 47.2, IEEE, 2001, S. 498–519. DOI: 10.1109/18.910572.

[26] Kuhn, Harold William: The Hungarian method for the assignment problem. In: *Naval research logistics quarterly* Vol. 2.1-2, Wiley Online Library, 1955, S. 83–97. DOI: 10.1002/nav.3800020109.

[27] Lengauer, Christian; Apel, Sven und Dörre, Jens: *MapReduceFoundation: Typing of MapReduce*. Universität Passau. URL: https://www.infosun.fim.uni-passau.de/cl/MapReduceFoundation, besucht am 13. 09. 2020.

[28] Liu, Xiaonan; Yin, Meijuan; Luo, Junyong und Chen, Wuping: An improved affinity propagation clustering algorithm for large-scale data sets. In: *2013 Ninth International Conference on Natural Computation (ICNC)*, IEEE, 2013, S. 894–899. DOI: 10.1109/ICNC.2013.6818103.

[29] Munkres, James: Algorithms for the assignment and transportation problems. In: *Journal of the society for industrial and applied mathematics* Vol. 5.1, SIAM, 1957, S. 32–38.

[30] Nentwig, Markus und Rahm, Erhard: Incremental clustering on linked data. In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2018, S. 531–538. DOI: 10.1109/ICDMW.2018.00084.

[31] Obraczka, Daniel; Saeedi, Alieh und Rahm, Erhard: Knowledge graph completion with FAMER. In: *Proceedings of the 1st KDD workshop on Data Integration for Knowledge Graphs (DI2KG)*, 2019.

[32] Pearl, Judea: Reverend bayes on inference engines: a distributed hierarchical approach. In: *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI Press, 1982, S. 133–136.

[33] Pedregosa, Fabian; Varoquaux, Gaël; Gramfort, Alexandre; Michel, Vincent; Thirion, Bertrand; Grisel, Olivier; Blondel, Mathieu; Prettenhofer, Peter; Weiss, Ron und Dubourg, Vincent u.a.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* Vol. 12, 2011, S. 2825–2830.

[34] Peukert, Eric: Cloud Data Management. Kapitel 1: Infrastruktur und Services. Vorlesungsskript. Institut für Informatik, Universität Leipzig, 2019. URL: `https://dbs.uni-leipzig.de/file/CDM_WS_2019_02_Infrastruktur_Service(Teil1)`, besucht am 29.08.2020.

[35] Peukert, Eric: Cloud Data Management. MapReduce Teil 1. Vorlesungsskript. Institut für Informatik, Universität Leipzig, 2019. URL: `https://dbs.uni-leipzig.de/file/CDM_WS_2019_04_MapReduce_Teil1.pdf`, besucht am 13.09.2020.

[36] Rahm, Erhard: Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung. 1. Auflage. Addison-Wesley-Verlag, 1994. ISBN: 3-89319-702-8.

[37] Saeedi, Alieh; Nentwig, Markus; Peukert, Eric und Rahm, Erhard: Scalable matching and clustering of entities with FAMER. In: *Complex Systems Informatics and Modeling Quarterly*, eISSN: 2255-9922, 2018, S. 61–83. DOI: `10.7250/csimq.2018-16.04`.

[38] Saeedi, Alieh; Peukert, Eric und Rahm, Erhard: Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In: *European Conference on Advances in Databases and Information Systems*, LNCS 10509, Springer, 2017, S. 278–293. DOI: `10.1007/978-3-319-66917-5_19`.

[39] Saeedi, Alieh; Peukert, Eric und Rahm, Erhard: Incremental Multi-source Entity Resolution for Knowledge Graph Completion. In: *European Semantic Web Conference*, Springer, 2020, S. 393–408. DOI: `10.1007/978-3-030-49461-2_23`.

[40] Saeedi, Alieh; Peukert, Eric und Rahm, Erhard: Using link features for entity clustering in knowledge graphs. In: *European Semantic Web Conference*, LNCS 10843, Springer, 2018, S. 576–592. DOI: 10.1007/978-3-319-93417-4_37.

[41] Schubert, Matthias: Datenbanken: Theorie, Entwurf und Programmierung relationaler Datenbanken. 2. bearbeitete Auflage. SpringerLink Bücher. Wiesbaden: Vieweg+Teubner Verlag, 2007. ISBN: 978-3-8351-9108-2.

[42] Smith, Taylor: *clust4j: Projekt-Repository auf GitHub*. URL: https://github.com/tgsmith61591/clust4j, besucht am 16. 09. 2020.

[43] Stern, Kevin: *software-and-algorithms: Projekt-Repository auf GitHub*. URL: https://github.com/KevinStern/software-and-algorithms, besucht am 16. 09. 2020.

[44] Tanner, Robert Michael: A recursive approach to low complexity codes. In: *IEEE Transactions on information theory* Vol. 27.5, IEEE, 1981, S. 533–547. DOI: 10.1109/TIT.1981.1056404.

[45] The Apache Software Foundation: *Component Stack*. URL: https://ci.apache.org/projects/flink/flink-docs-release-1.10/internals/components.html, besucht am 29. 08. 2020.

[46] The Apache Software Foundation: *Distributed Runtime Environment*. URL: https://ci.apache.org/projects/flink/flink-docs-release-1.10/concepts/runtime.html, besucht am 29. 08. 2020.

[47] The Apache Software Foundation: *Flink DataSet API Programming Guide*. URL: https://ci.apache.org/projects/flink/flink-docs-stable/dev/batch/, besucht am 17. 09. 2020.

[48] The Apache Software Foundation: *Gelly: Flink Graph API*. URL: https://ci.apache.org/projects/flink/flink-docs-stable/dev/libs/gelly, besucht am 14. 09. 2020.

[49] The Apache Software Foundation: *Iterative Graph Processing*. URL: https://ci.apache.org/projects/flink/flink-docs-stable/dev/libs/gelly/iterative_graph_processing.html, besucht am 14. 09. 2020.

[50] Wang, Kaijun; Zhang, Junying; Li, Dan; Zhang, Xinna und Guo, Tao: Adaptive affinity propagation clustering. In: *Acta Automatica Sinica* Vol. 33(12), 2007, S. 1242–1246.

[51] Zabaras, Nicholas: Probabilistic Graphical Models. Lecture 9: Belief Propagation, Max-Sum Algorithm, and Introduction to the Junction Tree Algorithm. Vorlesungsskript. University of Notre Dame, 2018. URL: `https://www.zabaras.com/probabilistic-graphical-models`, besucht am 14.08.2020.

Anhang

Anhang A

Detaillierte Ergebnisse zur Evaluation der Clustering-Qualität

Die folgende Anlage enthält detaillierte Darstellungen der Ergebnisse sämtlicher Experimente zur Evaluation der Clustering-Qualität in Abschnitt 6.4. Zunächst wird die für die Experimente verwendete Konfiguration der bereits in FAMER integrierten Clustering-Verfahren aufgeführt und erläutert. Anschließend folgen Abbildungen der Resultate zur Clustering-Qualität für die MS-Clean-ER Datensammlungen DS-G, DS-M und DS-P sowie für die MSCD-ER Datensammlung DS-C.

Tabelle A.1 zeigt die Parameterkonfiguration der sieben Clustering-Algorithmen, die mit (H)AP und MSCD-(H)AP bei der Evaluation der Clustering-Qualtität in Abschnitt 6.4 verglichen wurden. Dargestellt sind die in FAMER verwendeten Parameterbezeichnungen. *PrioritySelection* definiert für die Auswahl von Knoten entsprechend der Knotenpriorität, ob Knoten mit geringerer (MIN) oder höherer (MAX) Priotirät favorisiert werden. *DegreeCoef, simValueCoef* und *strengthCoef* stehen für die Koeffizienten des Kantengrads, des Kantengewichts und der Kantenstärke in CLIP (vgl. Abschnitt 3.1.4). *SourceNumber* definiert die Anzahl der Quellen für die zu verarbeitende Datensammlung. Der boolsche Parameter *removeSourceConsistentVertices* legt fest, ob clean-source-konsistente Komponenten zu Beginn der zweiten Phase von CLIP aus dem Graphen entfernt (true) oder weiter verarbeitet werden (false). Der Parameter *delta* erlaubt es CLIP, bei der Ermittlung von *Strong-Links* und *Normal-Links* nicht nur die maximalen Links zu berücksichtigen. Anstatt dessen werden alle Kanten eines Knotens zu allen anderen Knoten einer gemeinsamen Quelle berücksichtigt, deren Kantengewicht maximal um einen Wert von *delta* vom Gewicht des maximalen Links zu der Quelle abweicht. Der Parameter *epsilon* von CCPivot wird in Abschnitt 3.1.4 erläutert. Die Konfiguration von (H)AP und MSCD-(H)AP ist in Abschnitt 6.2 beschrieben.

Algorithmus	Parameter	Wert
CCPivot	epsilon	0,9
Center	prioritySelection	MIN
CLIP	degreeCoef	0,2
	delta	0,0
	removeSourceConsistentVertices	false
	simValueCoef	0,5
	sourceNumber	DS-G: 4 DS-M: 5 DS-P: 5 DS-C: 23
	strengthCoef	0,3
Connected Components	-	-
Merge-Center	prioritySelection	MIN
Star-1	prioritySelection	MIN
Star-2	prioritySelection	MIN

Tabelle A.1: Parameterkonfiguration der Clustering-Algorithmen in FAMER für die vergleichende Evaluation der Clustering-Qualität

Die Abbildungen A.1 bis A.12 zeigen die Genauigkeiten, Trefferquoten und die Resultate im F-Maß der Clustering-Algorithmen für die verschiedenen Datensammlungen und Teilkorpora. Dargestellt sind die Durchschnittswerte aus zehn (DS-G, DS-M und DS-C) bzw. fünf (DS-P) Versuchsausführungen. Die Fehlerbalken geben Auskunft über die Standardabweichung der Messwerte.

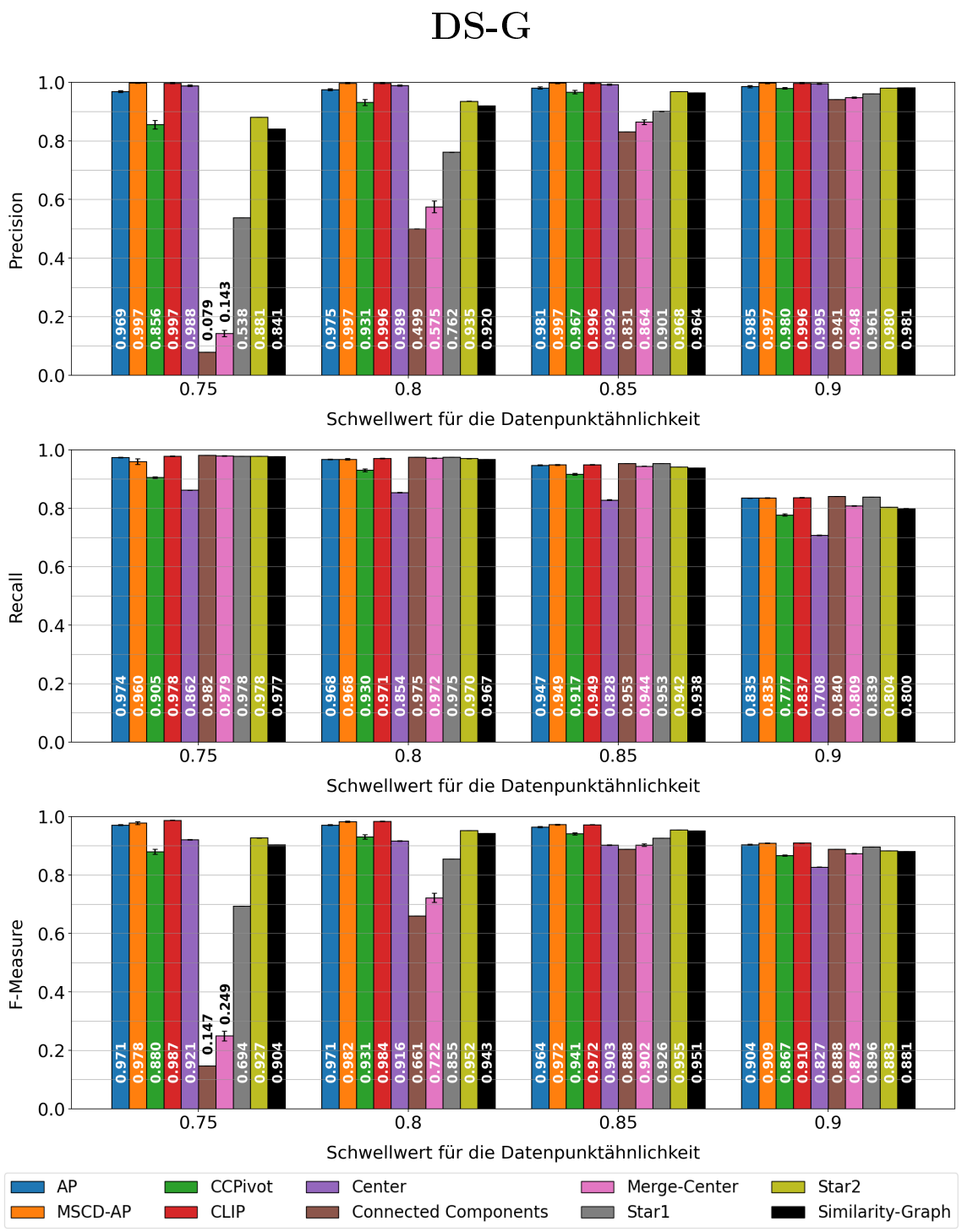


Abbildung A.1: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering der Datensammlung DS-G

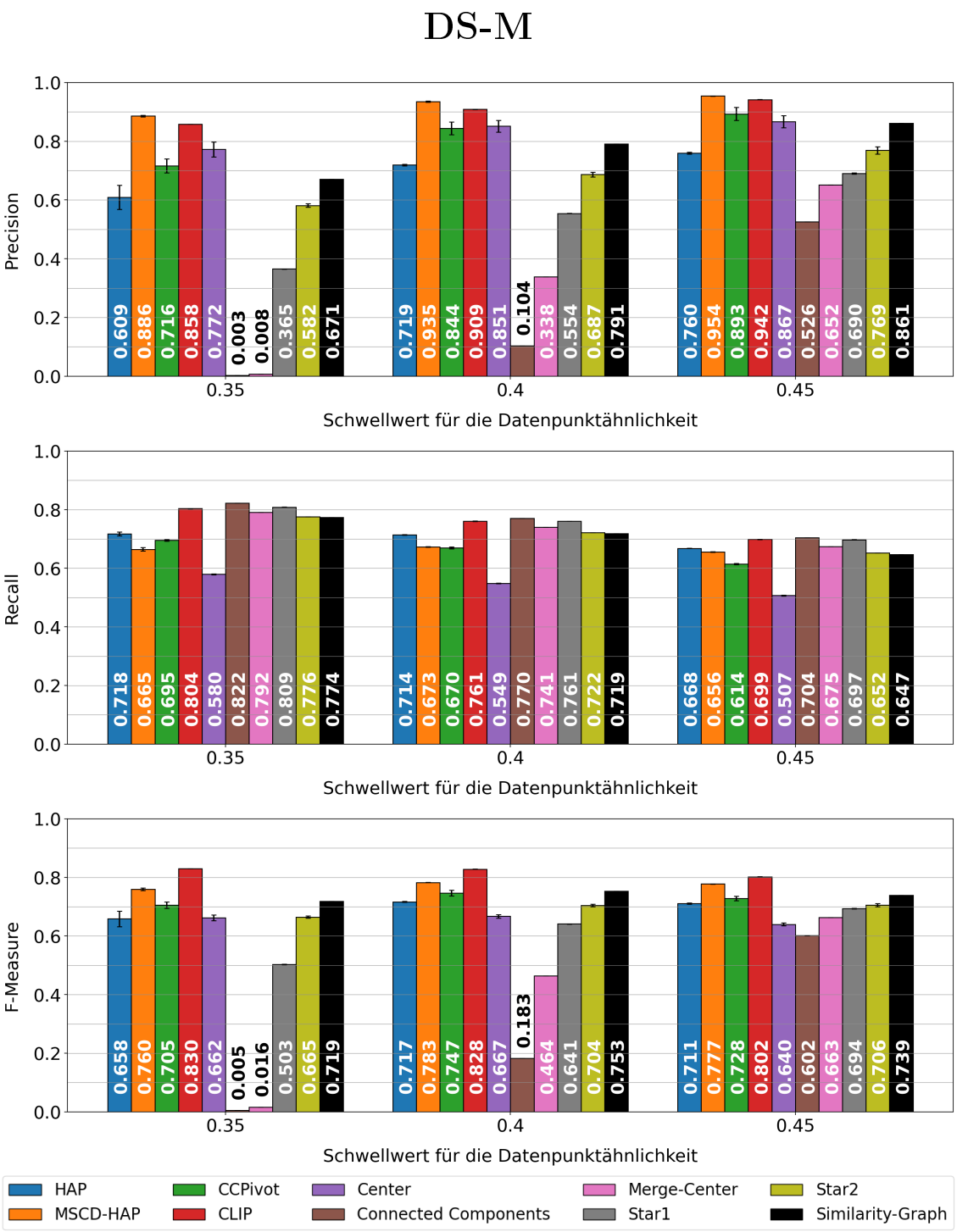


Abbildung A.2: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering der Datensammlung DS-M

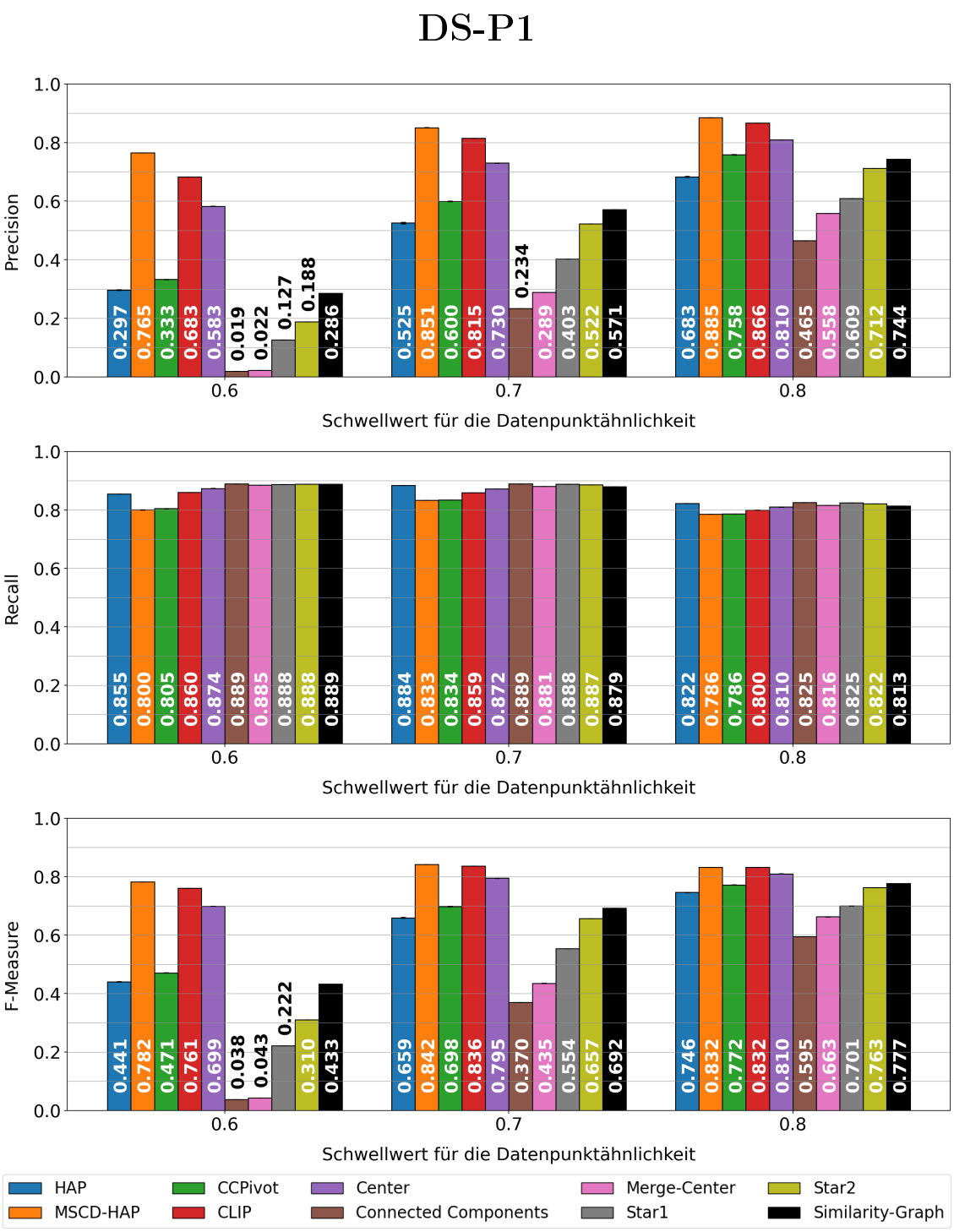


Abbildung A.3: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering der Datensammlung DS-P1

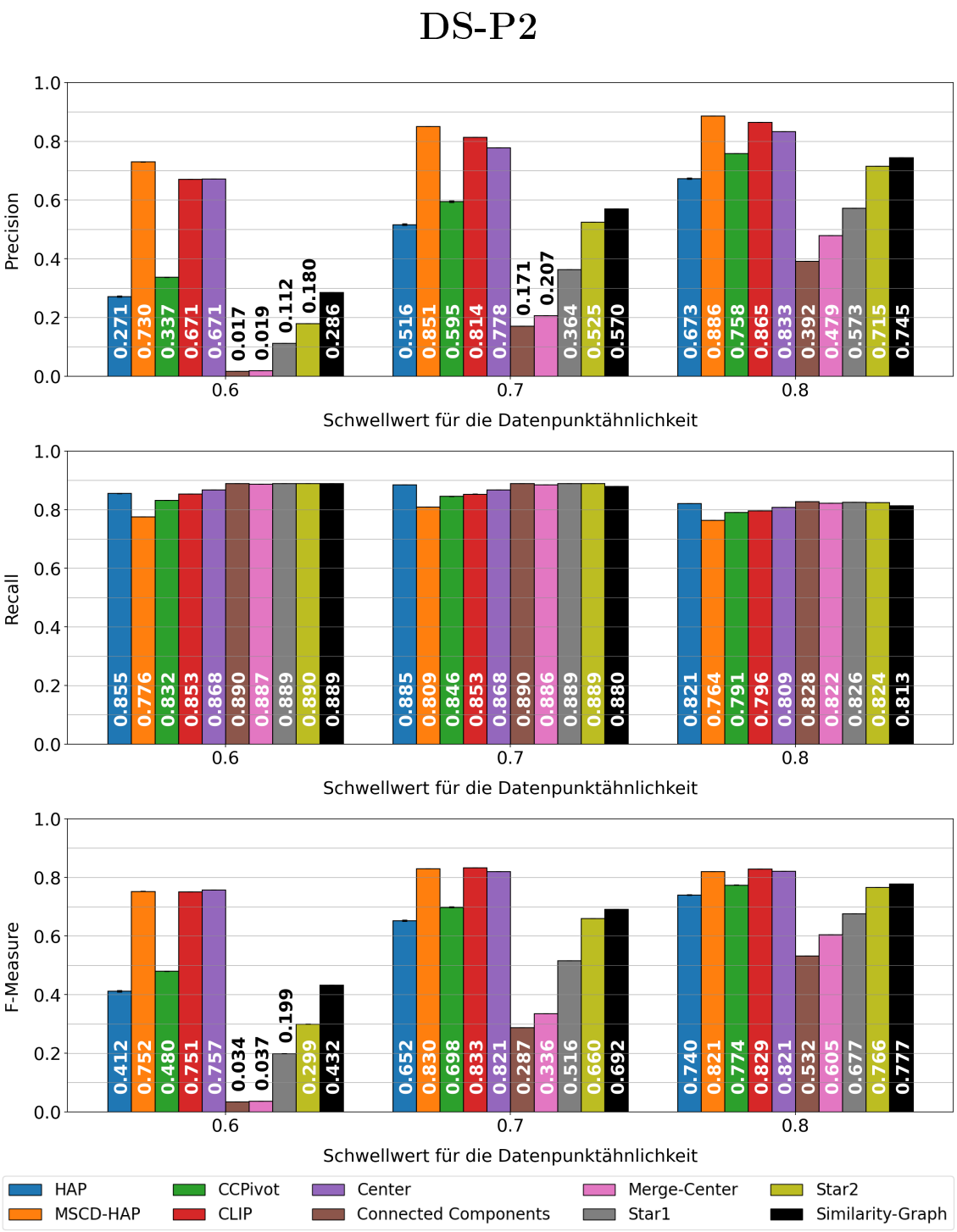


Abbildung A.4: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering der Datensammlung DS-P2

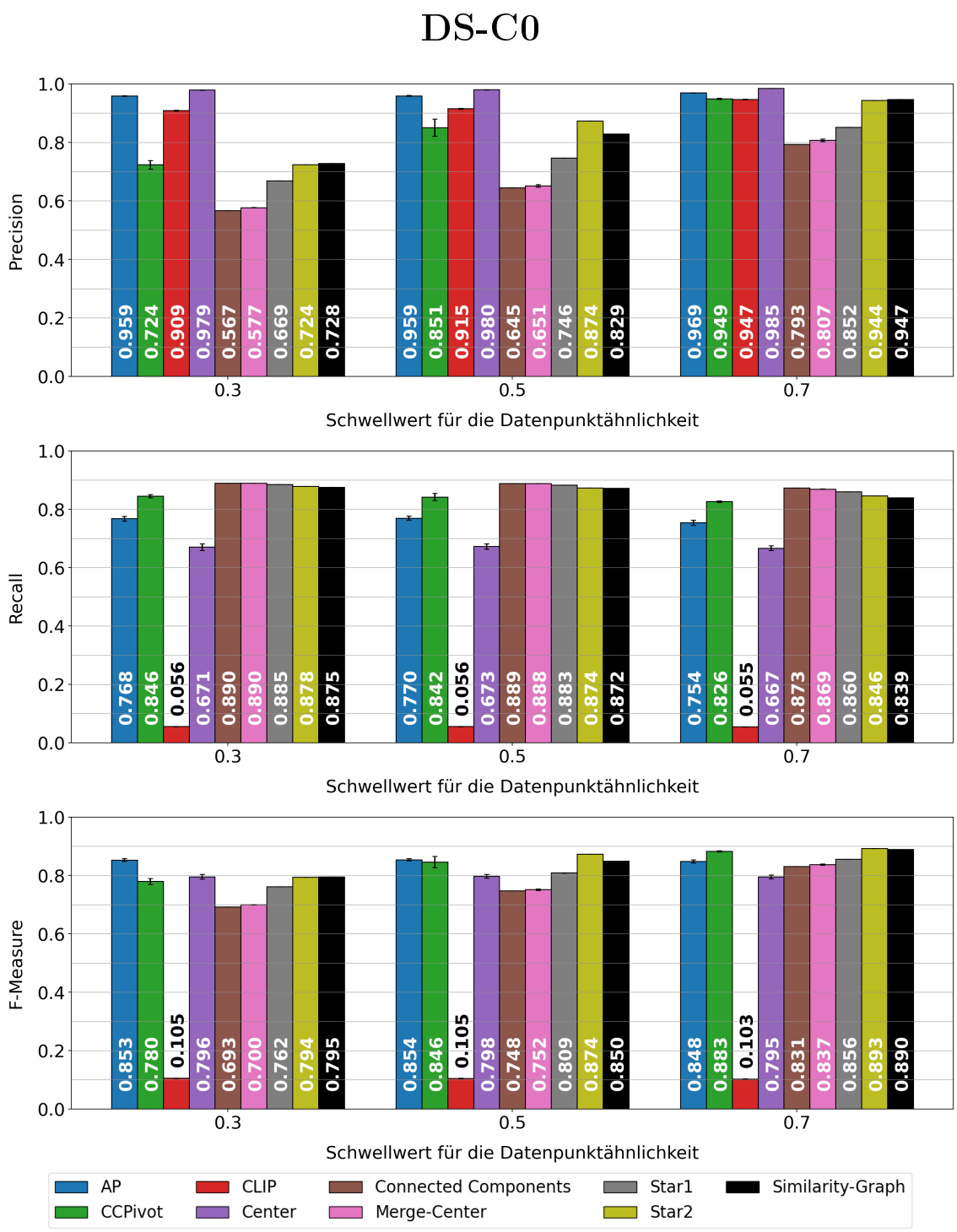


Abbildung A.5: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C0

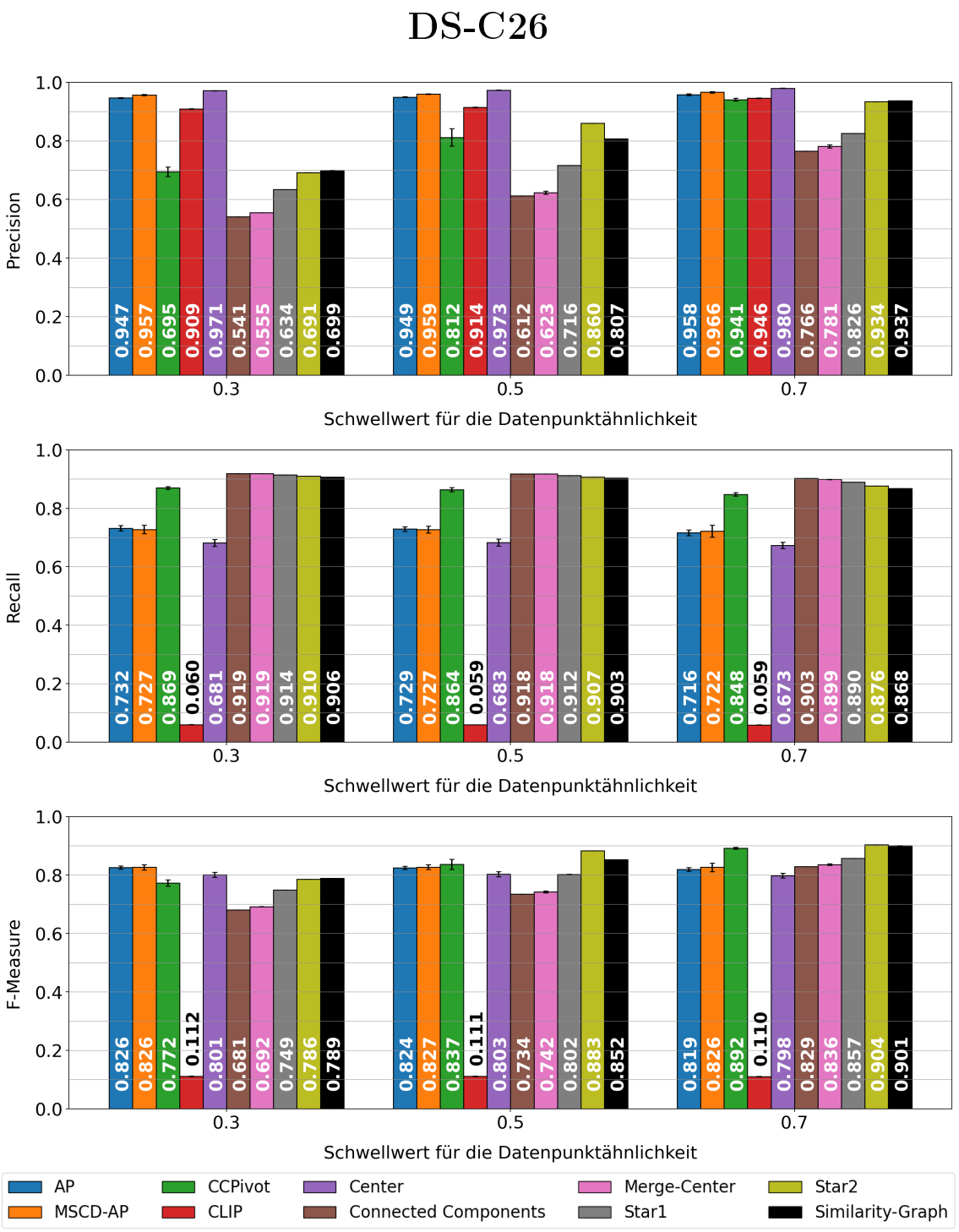


Abbildung A.6: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C26

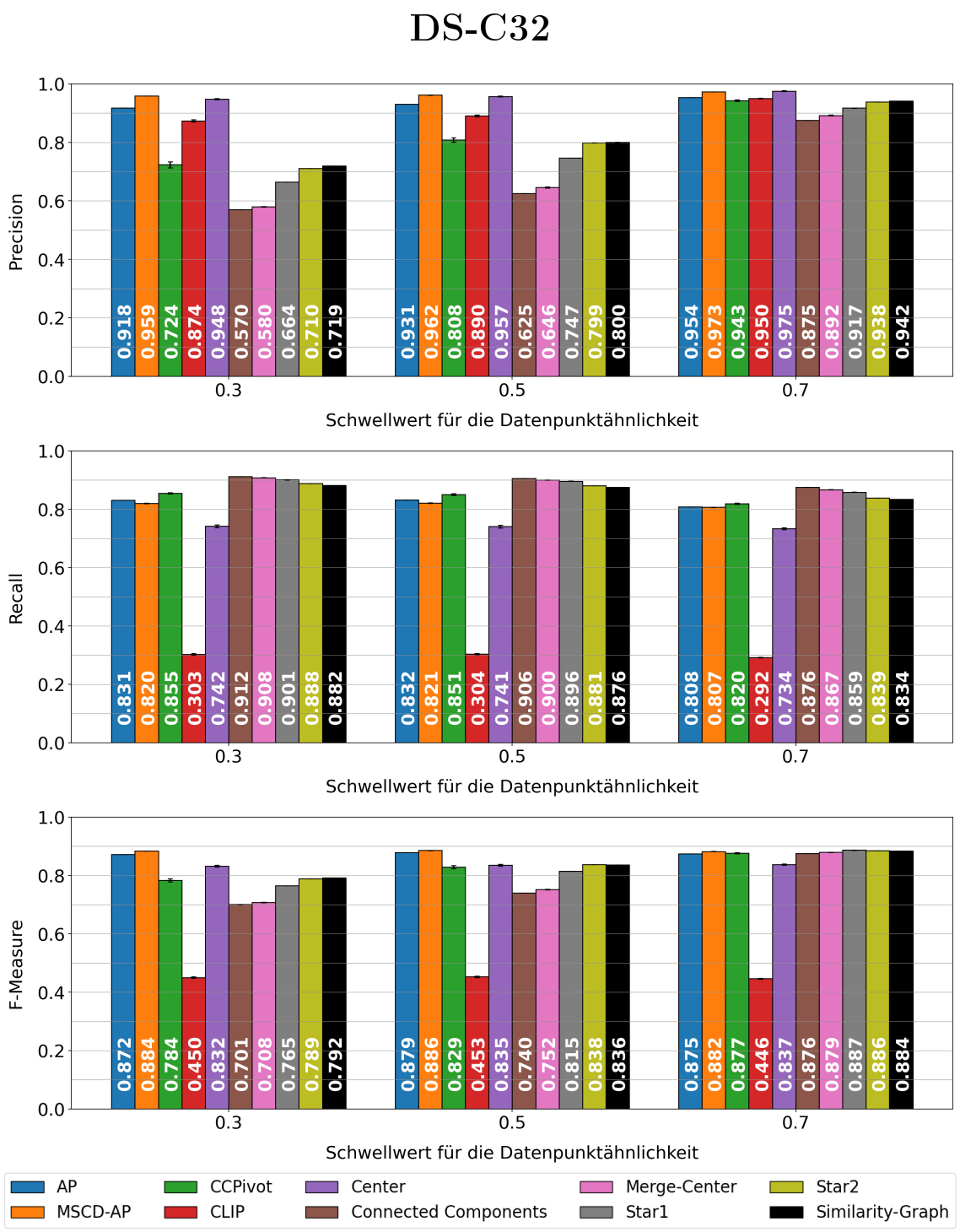


Abbildung A.7: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C32

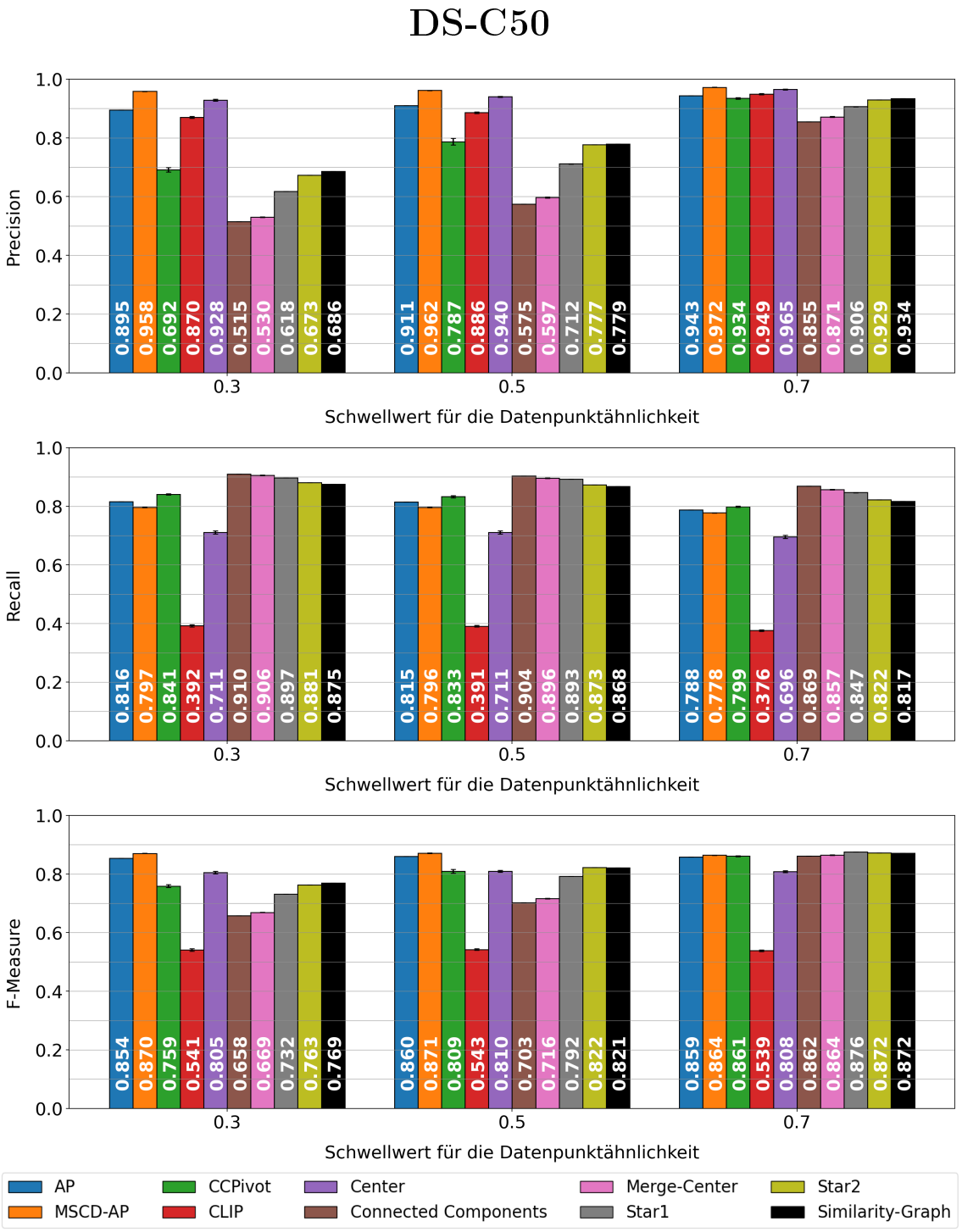


Abbildung A.8: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C50

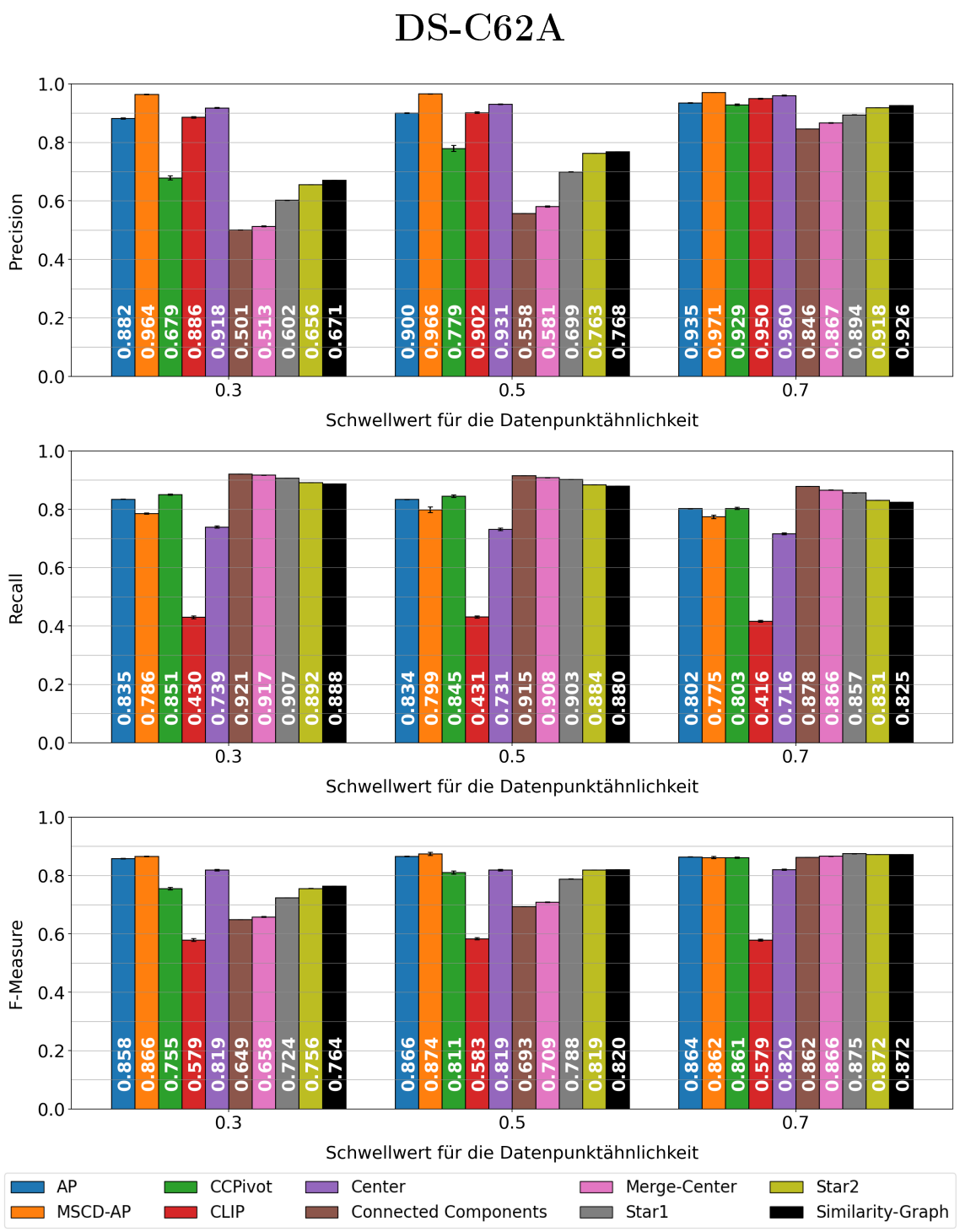


Abbildung A.9: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C62A

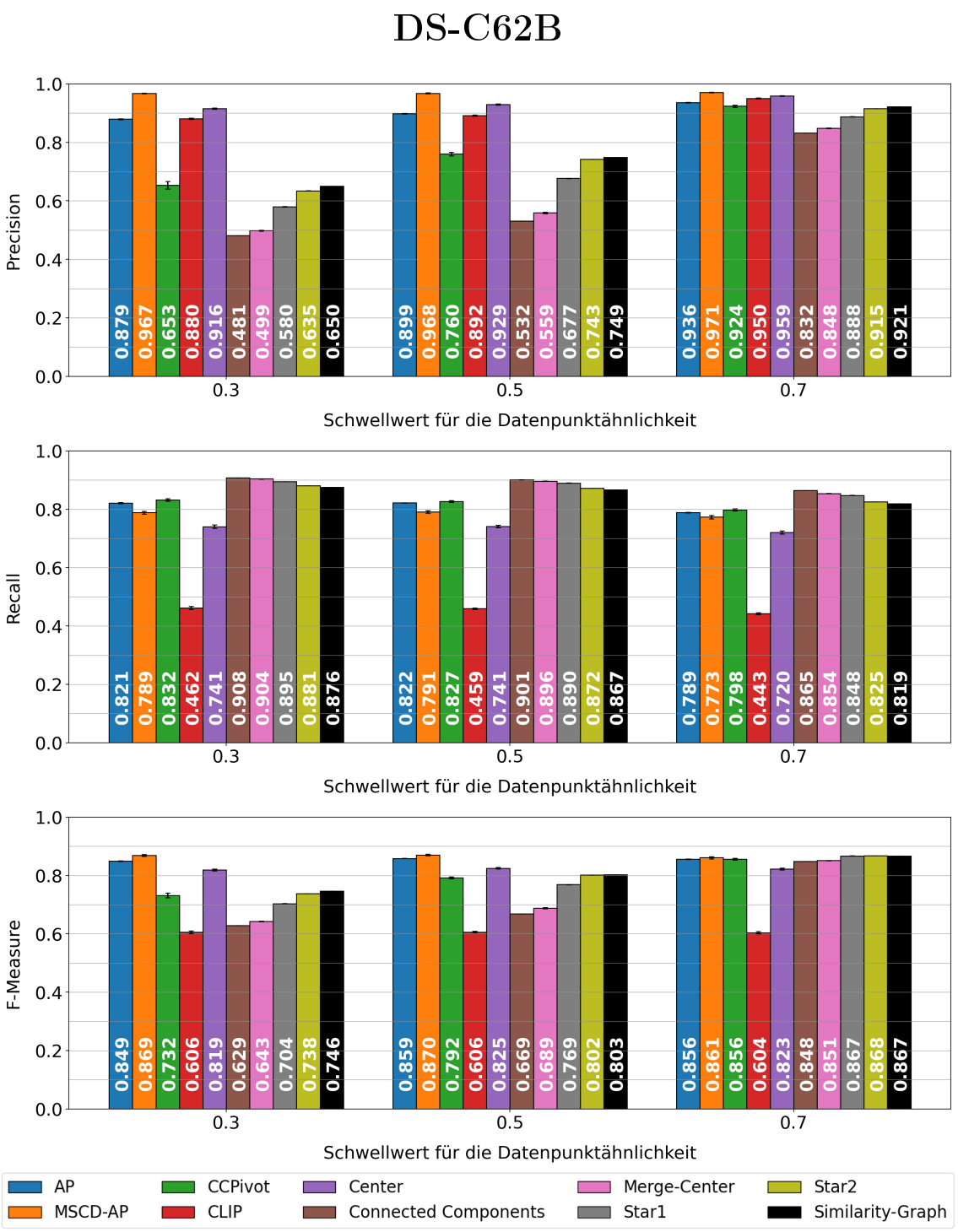


Abbildung A.10: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C62B

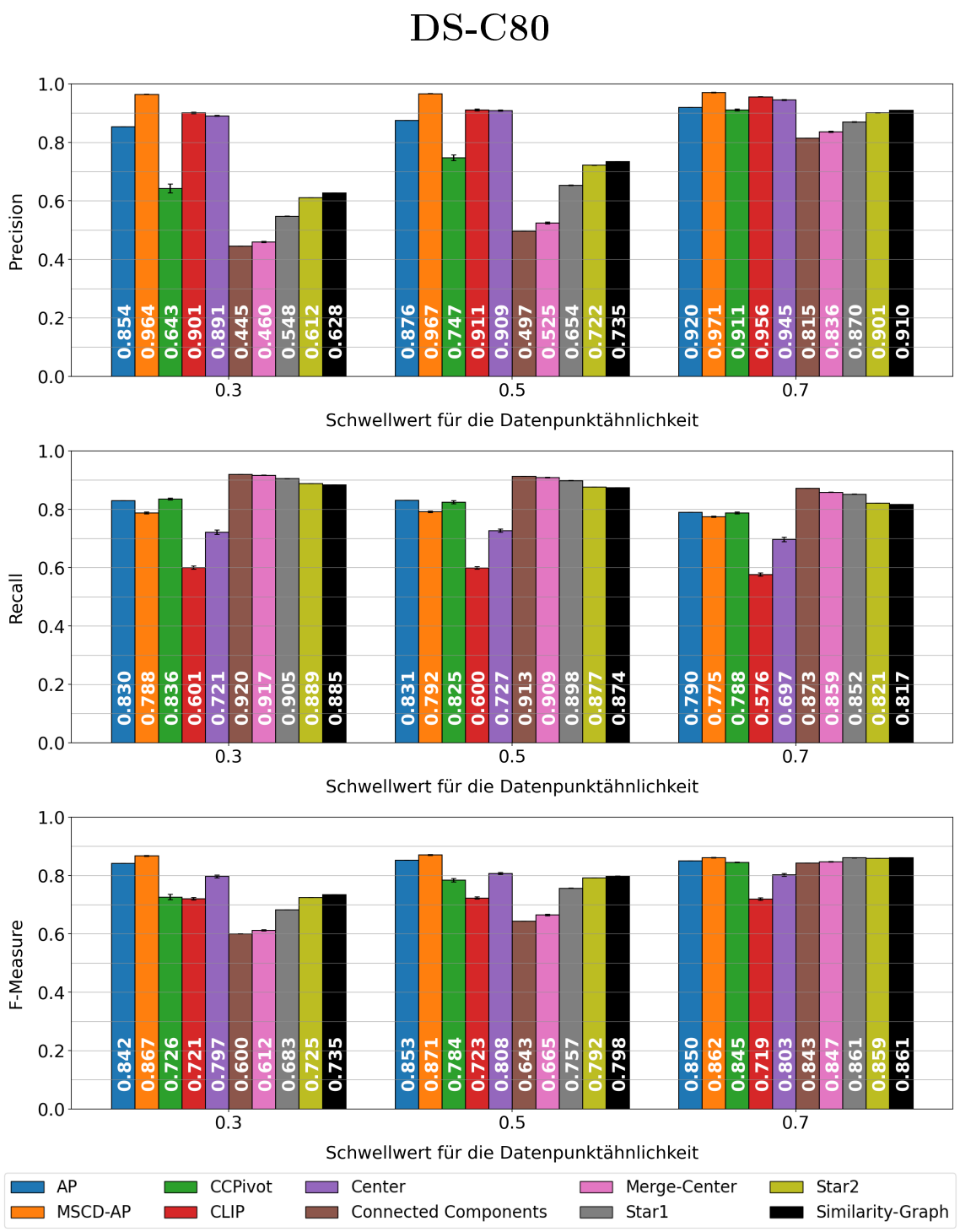


Abbildung A.11: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C80

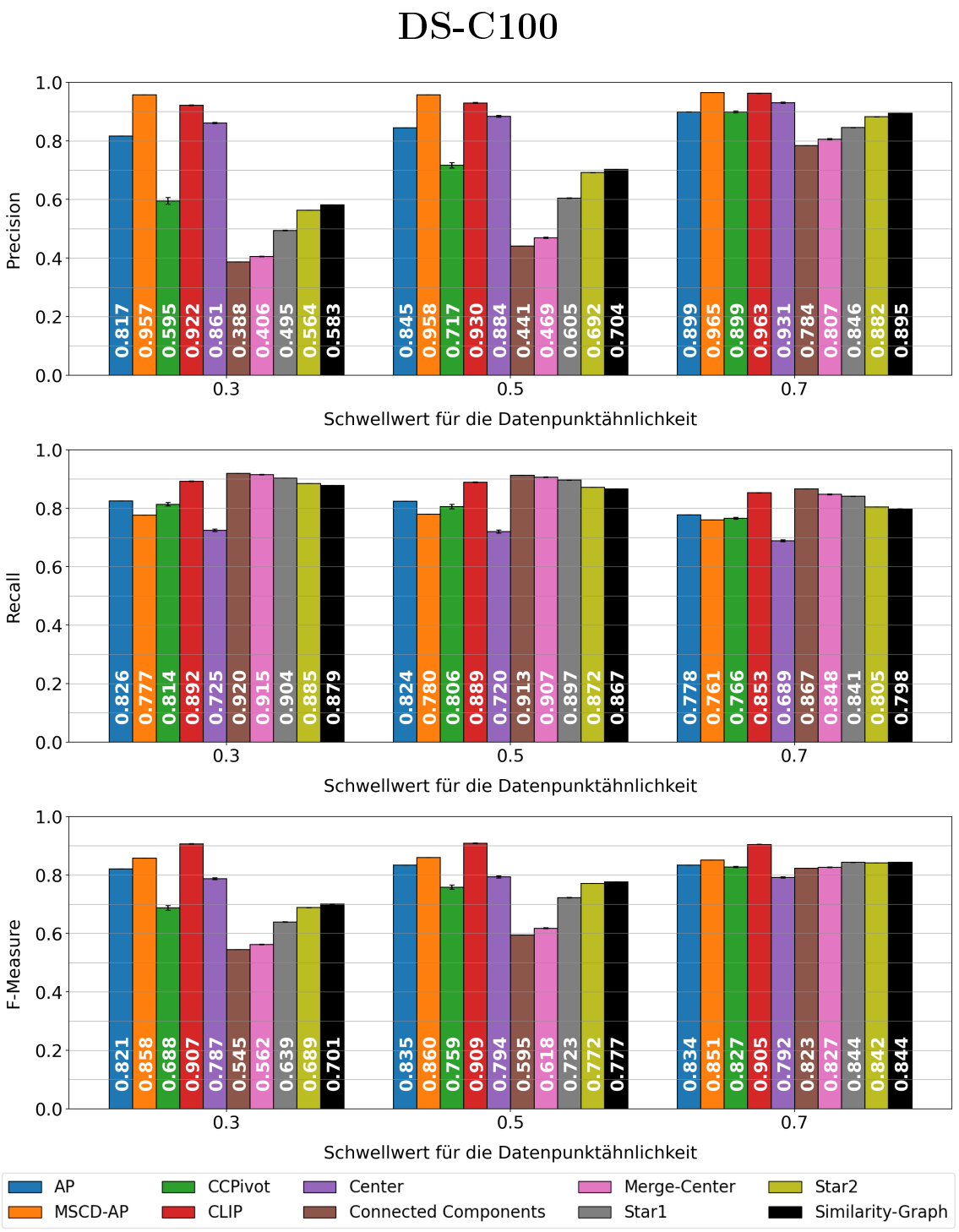


Abbildung A.12: ER-Ergebnisqualität der verschiedenen Algorithmen für das Clustering des Teilkorpus DS-C100

Anhang B

Detaillierte Ergebnisse zur Laufzeitevaluation

Im Folgenden werden die Ergebnisse zweier Experimente zur Laufzeitevaluation in Abschnitt 6.5 in detaillierter Form dargestellt. Abbildung B.1 zeigt die Laufzeiten der verglichenen Algorithmen für das Clustering von DS-P. Abbildung B.2 gibt sowohl die Laufzeiten als auch die Ergebnisqualitäten des Clusterings von DS-P mit MSCD-HAP wieder, welche dieses mit unterschiedlichen Einstellungen für die maximale Partitionsgröße erzielt. Die Messungen erfolgten auf dem in Abschnitt 6.2 beschriebenen Flink-Cluster. Abgebildet sind die Durchschnittswerte aus fünf Versuchsausführungen. Die Fehlerbalken geben Auskunft über die Standardabweichung der Messwerte.

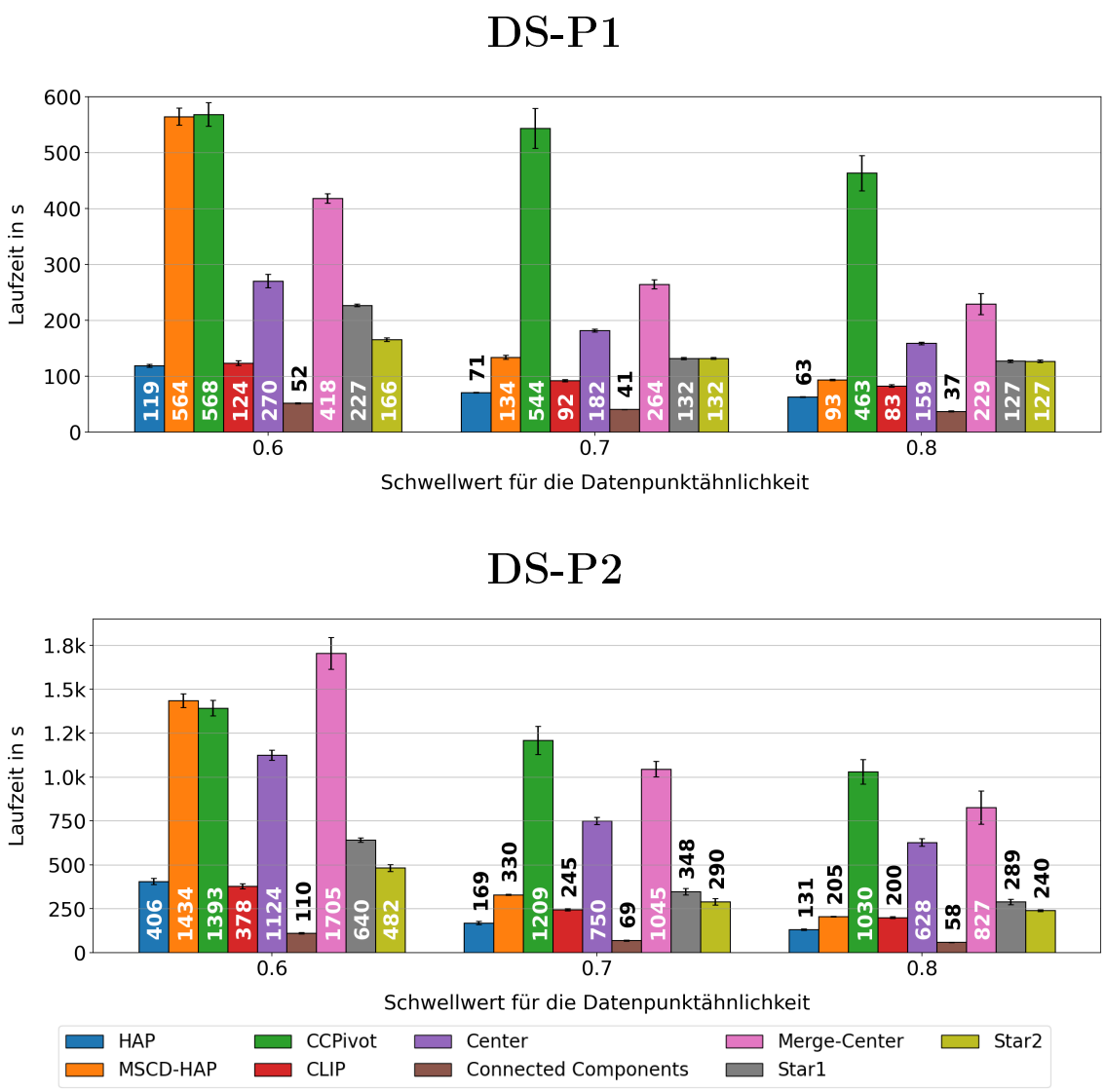


Abbildung B.1: Laufzeit der verschiedenen Algorithmen für das Clustering der Datensammlungen DS-P1 und DS-P2

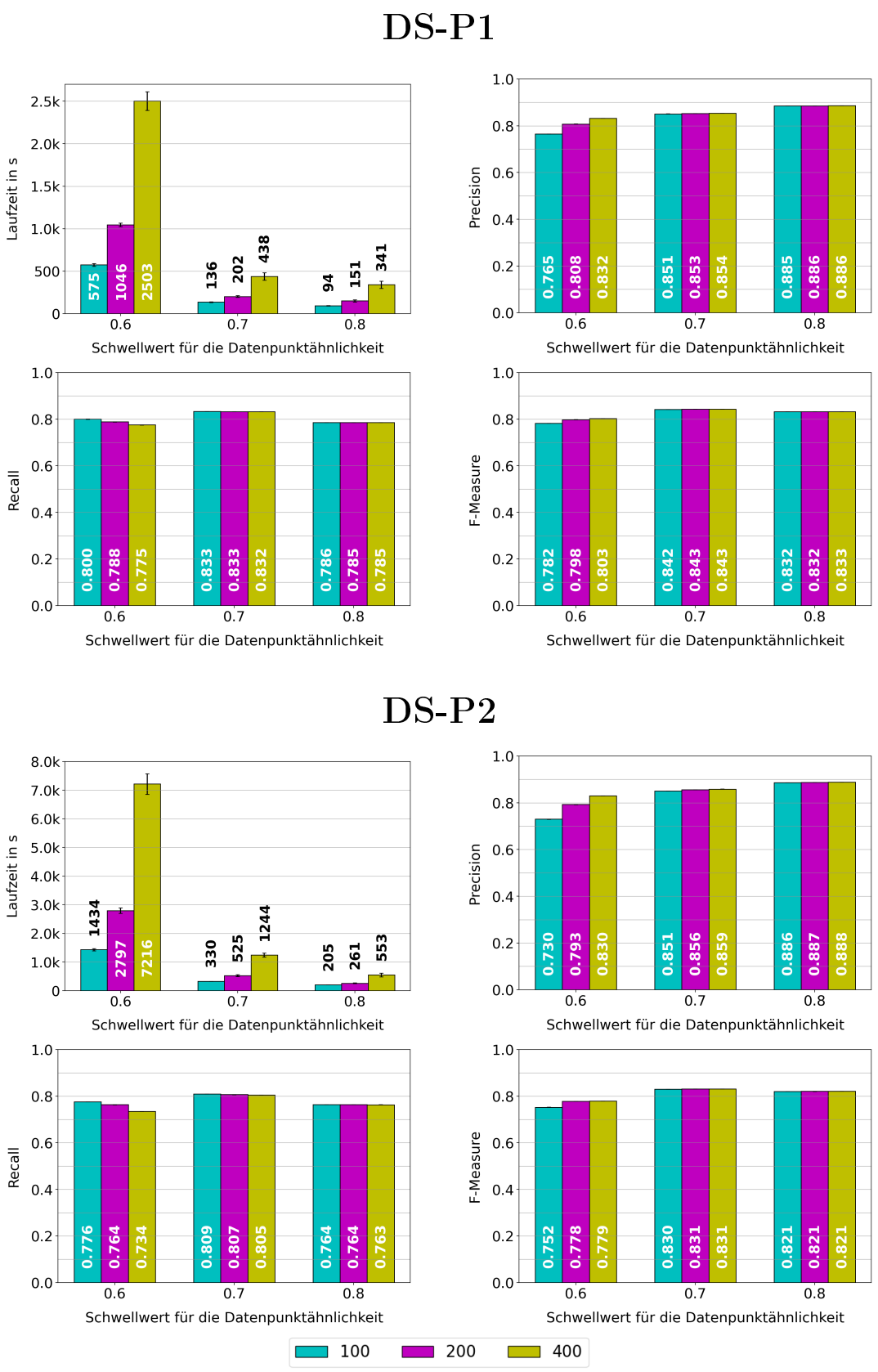


Abbildung B.2: Laufzeit und ER-Ergebnisqualität von MSCD-HAP für das Clustering der Datensammlungen DS-P1 und DS-P2 bei verschiedenen Einstellungen für die maximale Partitionsgröße

Anhang C

Detaillierte Ergebnisse zur Variantenbetrachtung von MSCD-HAP

Die folgende Anlage enthält eine detaillierte Darstellung der Ergebnisse der Experimente zur Variantenbetrachtung von MSCD-HAP in Abschnitt 6.6. Die Abbildungen C.1 bis C.4 zeigen sowohl die Laufzeiten als auch die Ergebnisqualitäten des Clusterings von DS-P für HAP, MSCD-HAP und zwei Mischtypen der beiden Algorithmen. Abb. C.2 und Abb. C.4 geben die in Abschnitt 6.6 vorgestellten Resultate für die Partitionsgröße 400 wieder. Zusätzlich dazu führen Abb. C.1 und Abb. C.3 die Ergebnisse für die Partitionsgröße 100 auf. Die Messungen erfolgten auf dem in Abschnitt 6.2 beschriebenen Flink-Cluster. Abgebildet sind die Durchschnittswerte aus fünf Versuchsausführungen. Die Fehlerbalken geben Auskunft über die Standardabweichung der Messwerte.

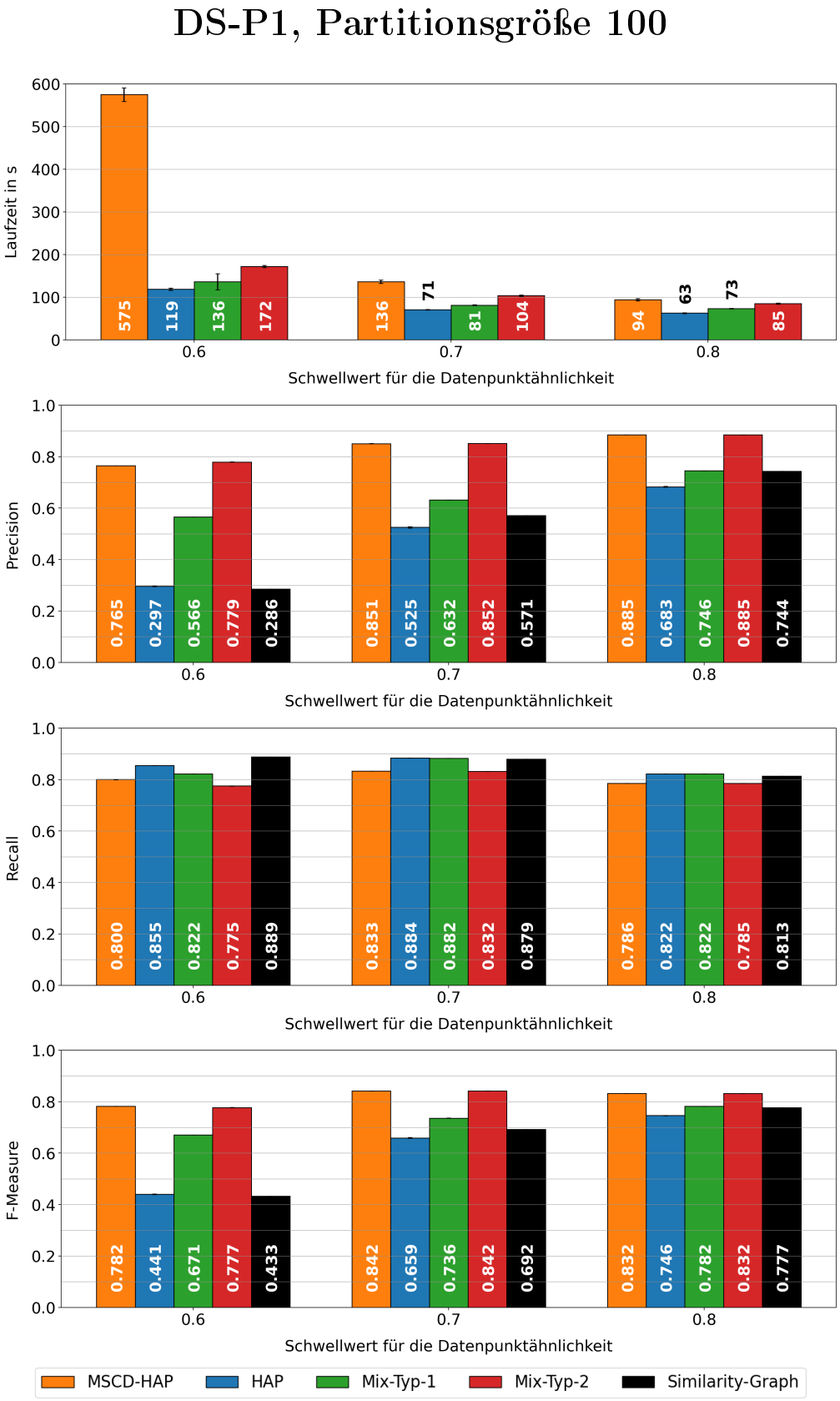


Abbildung C.1: Laufzeit und ER-Ergebnisqualität der verschiedenen Mischvarianten aus HAP und MSCD-HAP für das Clustering der Datensammlung DS-P1 mit einer maximalen Partitionsgröße von 100

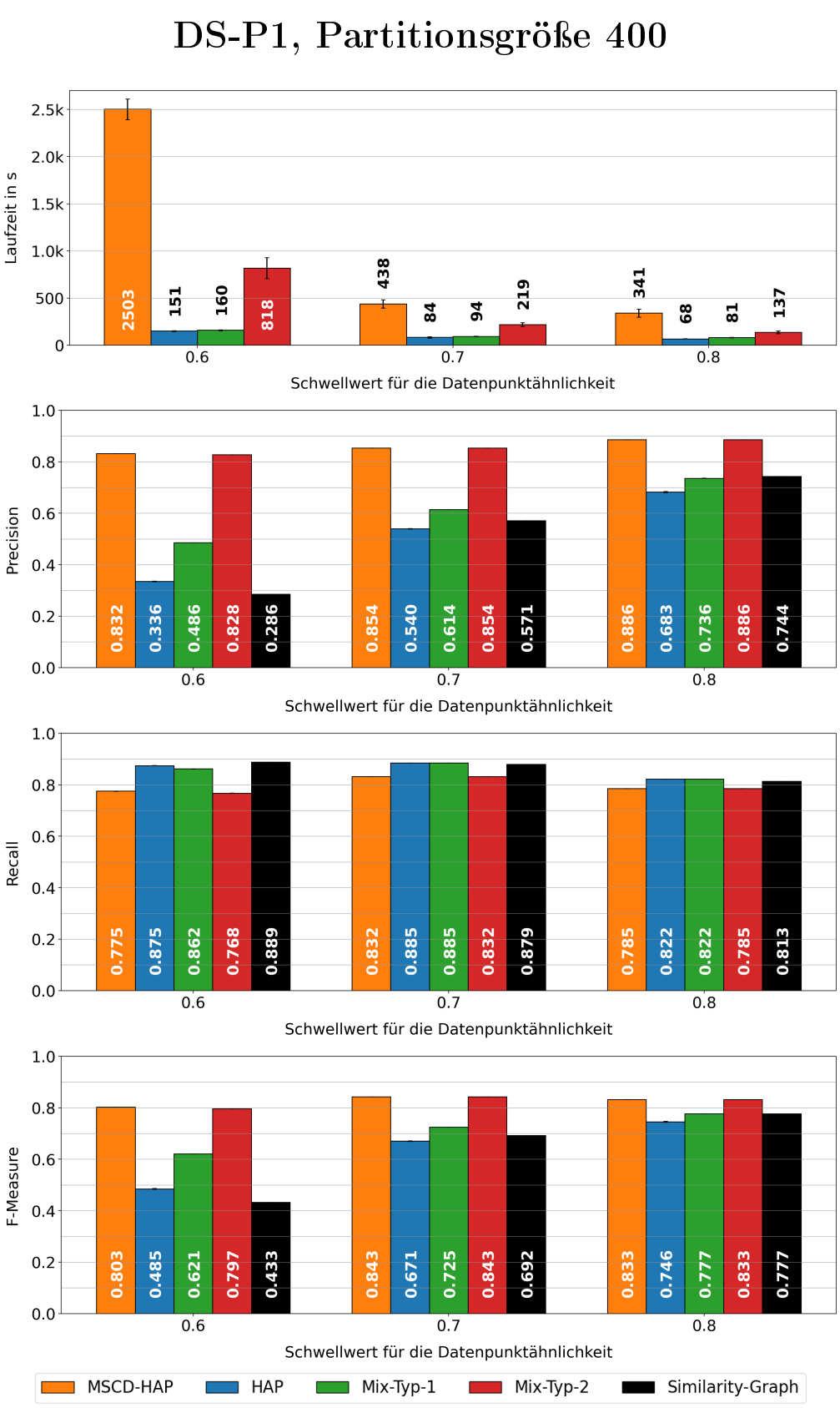


Abbildung C.2: Laufzeit und ER-Ergebnisqualität der verschiedenen Mischvarianten aus HAP und MSCD-HAP für das Clustering der Datensammlung DS-P1 mit einer maximalen Partitionsgröße von 400

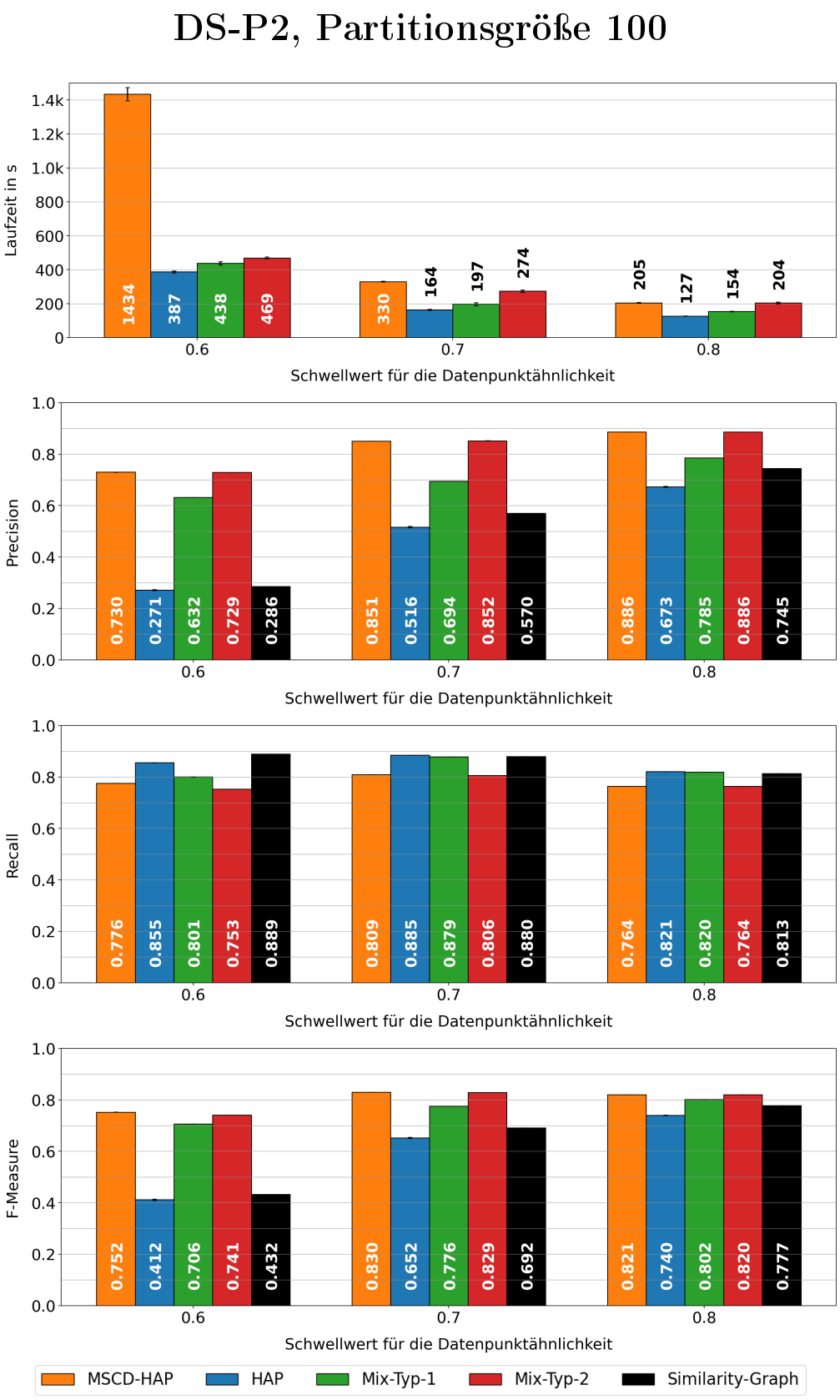


Abbildung C.3: Laufzeit und ER-Ergebnisqualität der verschiedenen Mischvarianten aus HAP und MSCD-HAP für das Clustering der Datensammlung DS-P2 mit einer maximalen Partitionsgröße von 100

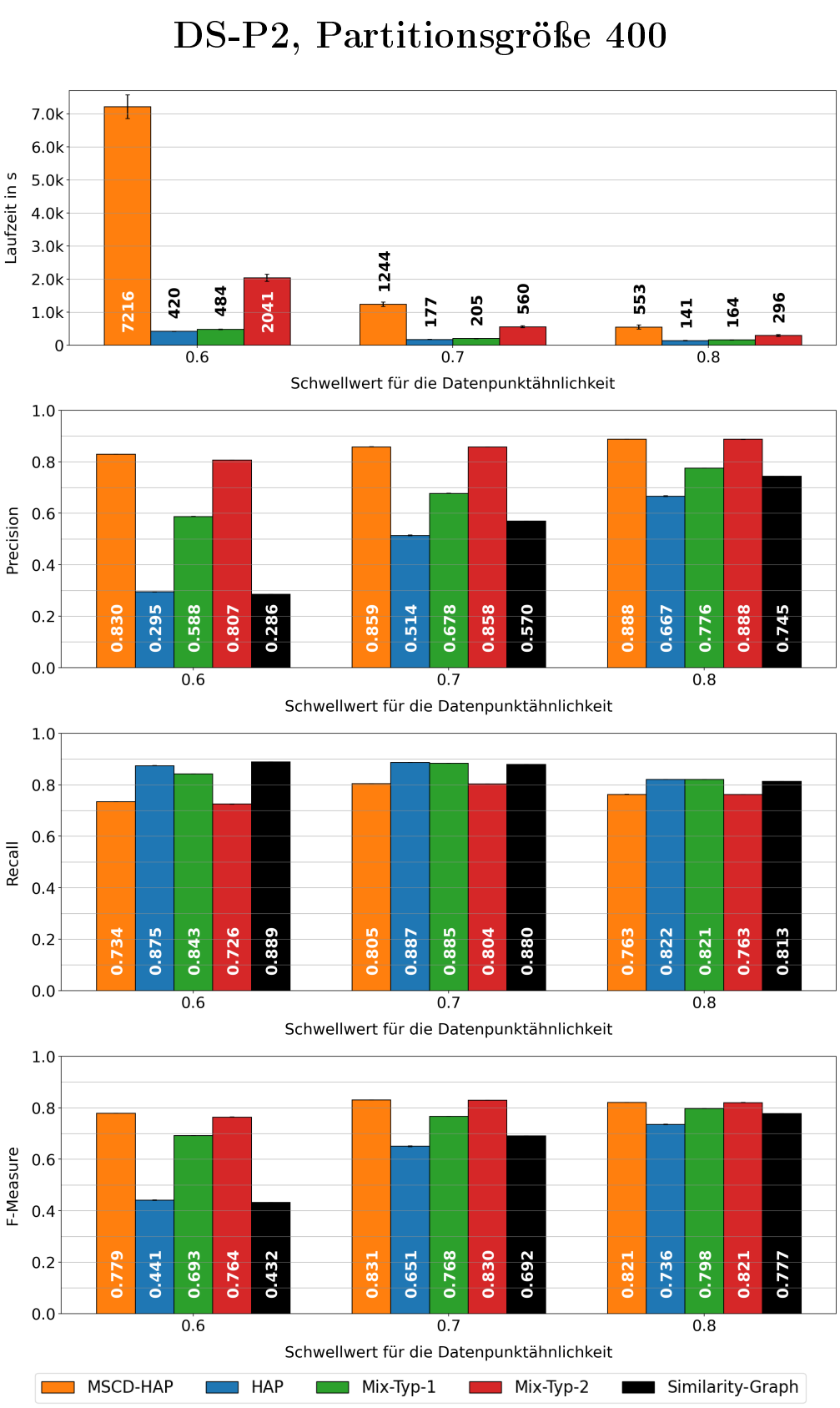


Abbildung C.4: Laufzeit und ER-Ergebnisqualität der verschiedenen Mischvarianten aus HAP und MSCD-HAP für das Clustering der Datensammlung DS-P2 mit einer maximalen Partitionsgröße von 400

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

Ort, Datum	Unterschrift