Universität Leipzig
Fakultät für Mathematik und Informatik
(Institut für Informatik)

Masterarbeit

Evaluierung von Blocking-Verfahren zum effizienten Matching großer Ontologien in den Lebenswissenschaften

Leipzig, 27.02.2014

vorgelegt von:
Anastasiya Chyhir
Studiengang: Informatik (Master)

Betreuer:
Prof. Dr. Erhard Rahm
Anika Groß
Universität Leipzig, Institut für Informatik, Abteilung Datenbanken

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Ontologien werden eingesetzt, um eine einheitliche und strukturierte Repräsentation von Wissen zu gewährleisten. Sie stellen einen gerichteten Graphen dar, dessen Knoten und Kanten jeweils die Konzepte und Beziehungen zwischen den Konzepten repräsentieren. Ontologien und deren Anwendung haben insbesondere in den Lebenswissenschaften zunehmend an Bedeutung gewonnen [3]. Dort werden sie in verschiedenen Domänen wie z. B. der Anatomie, Chemie oder Molekularbiologie eingesetzt. Häufig annotieren die Ontologien realer Objekte wie Gene, Literaturquellen oder elektronische Patientenakten. In den Lebenswissenschaften existieren zahlreiche große Ontologien wie NCI Thesaurus (*The National Cancer Institute Thesaurus*) mit über 90.000 Konzepten oder SNOMED CT (*Systematized Nomenclature of Human and Veterinary Medicine*) mit über 300.000[1] Konzepten.

Innerhalb einer Domäne existieren häufig verschiedene Ontologien, welche ähnliches Wissen beschreiben. Beispielweise stellt das OBO Konsortium[2] mehr als 30 Ontologien aus dem Bereich der Anatomie Domäne zur Verfügung. Zwischen den (teilweise) überlappenden Ontologien müssen somit Mappings bestimmt werden. Die Menge der Korrespondenzen zwischen zwei Ontologien wird als Ontologie-Mapping bezeichnet. Ontologie-Mappings werden für die Datenintegration und den Austausch zwischen Anwendungen benötigt oder für das Merging von mehreren Ontologien [22] zu einer integrierten Ontologie wie z. B. Uberon [18] verwendet. Eine manuelle Bestimmung ist jedoch oft zu aufwendig oder nicht möglich, da die Ontologien teilweise sehr groß sind. Daher kommen häufig semi-automatische Match-Verfahren zum Einsatz [6, 21], um die Korrespondenzen zwischen den Konzepten verschiedener Ontologien zu bestimmen.

Das Matching von Ontologien sollte möglichst effektiv und effizient sein. Um Mappings von hoher Qualität zu bestimmen, ist eine korrekte und vollständige Identifikation von

[1] http://www.ihtsdo.org/snomed-ct/snomed-ct0/ Stand: 24.01.2014.
[2] *Open Biomedical Ontologies: http://www.obofoundry.org/.*

semantischen Korrespondenzen erforderlich. Dies ist insbesondere für sehr große Ontologien schwierig. Häufig führen Match-Systeme die Berechnung des kartesischen Produkts zwischen Ontologien im Hauptspeicher durch, was die Anwendbarkeit für sehr große Ontologien stark begrenzt. Weiterhin ist die Auswertung großer Suchräume zeitaufwendig, insbesondere wenn mehrere Matcher ausgeführt und deren Ergebnisse kombiniert werden müssen [22]. Die Ontology Alignment Evaluation Initiative (OAEI[3]) hat die Beurteilung der Stärken und Schwächen verschiedener Matching-Systeme, den Vergleich der Leistung von Techniken sowie die Verbesserung von Bewertungsmethoden zum Ziel. Viele Match-Systeme, die an der OAEI 2012 teilgenommen haben, waren nicht in der Lage, den *Large BioMed Track* (Matching von sehr großen biomedizinischen Ontologien) zu absolvieren. GOMMA Framework hat im OAEI 2012 [8] die besten F-Measure-Werte zwischen allen teilnehmenden Match-Systemen gezeigt. GOMMA wurde von der Abteilung Datenbanken der Universität Leipzig entwickelt und in dieser Arbeit verwendet (siehe Kapitel 2.4).

Es existieren verschiedene Verfahren, die ein effizientes Matching von Ontologien zum Ziel haben. In [21] werden Ansätze wie das frühzeitige Verwerfen von Korrespondenzen zwischen sehr unähnlichen Konzepten (*Early pruning*), paralleles Ontologie-Matching sowie die Wiederverwendung existierender Match-Ergebnisse betrachtet. Diese werden in Kapitel 2.5 näher diskutiert. Eine weitere Möglichkeit, große Ontologien effizient zu vergleichen, ist die Reduktion des Suchraums durch die Verwendung sogenannter Blocking-Methoden.

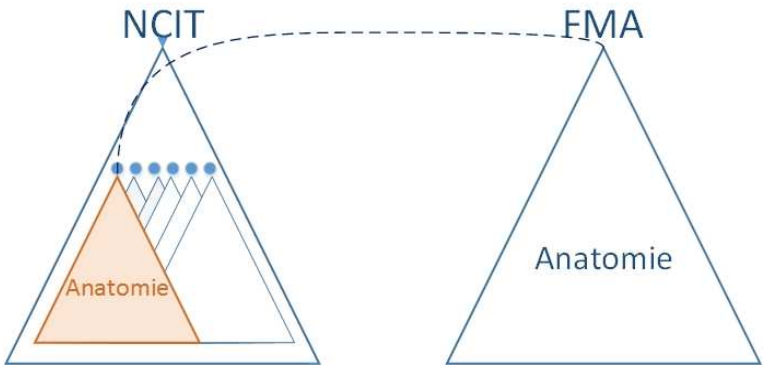


Abbildung 1.1-1 NCIT – FMA

[3] Ontology Alignment Evaluation Initiative http://oaei.ontologymatching.org/.

In Abbildung 1.1-1 ist das Matchproblem NCIT - FMA[4] dargestellt. NCIT enthält das Wissen zu 20 verschiedenen Teildomänen aus dem Bereich der Krebsforschung. Nur ein Teil des NCIT umfasst anatomisches Wissen. Im Gegensatz dazu beschreibt FMA ausschließlich die Anatomie-Domäne. Ein mögliches Blocking-Verfahren kann den Anatomieteil des NCIT automatisch bestimmen, so dass nur dieser Bereich mit der gesamten FMA abgeglichen werden muss. Dadurch können unnötige Vergleiche, die nicht zu sinnvollen Korrespondenzen führen, eingespart werden.

[4] *Foundational Model of Anatomy.*

1.2 Ziele der Arbeit

Das Hauptziel dieser Arbeit ist die Konzeption und Entwicklung von Blocking-Verfahren. Diese sollen anhand ausgewählter Matchprobleme in den Lebenswissenschaften vergleichend analysiert werden. Dabei kommen verschiedene Fragenstellungen auf: Wie stark reduzieren verschiedene Blocking-Verfahren den Suchraum? Führt die Verwendung von Blocking-Verfahren zu einem effizienteren Matching im Vergleich zum vollständigen Abgleich der Ontologien? Werden dabei Mappings von hoher Qualität bestimmt? Um diese Fragen zu beantworten, werden im Rahmen dieser Arbeit folgende Algorithmen entwickelt bzw. existierende Verfahren umgesetzt:

- Blocking von Ontologie-Subgraphen (*OntoSubBlocking*) ist ein Ansatz, der in [8] beschrieben wurde und in Kapitel 3.1 näher betrachtet wird. Dies ist ein defensiver Ansatz, der auf dem initialen Mapping basiert. Es werden die Teile der Ontologie (d. h. einige Subgraphen) ausgewählt, die besonders viele Korrespondenzen aus dem initialen Mapping haben.

- Blocking anhand zusammenhängender Ontologie-Subgraphen (*RelatedSub-GraphBlocking*) berücksichtigt die Verknüpfungen zwischen den Eingangsontologien, die mit Hilfe des initialen Mappings ermittelt werden, um nur die relevanten Ontologie-Teile für den weiteren Abgleich zu verwenden. Dieses Verfahren wurde von der Abteilung Datenbanken der Universität Leipzig entwickelt und im Rahmen dieser Arbeit umgesetzt (siehe Kapitel 3.2).

- Blocking anhand der am besten verknüpften Ontologie-Subgraphen (*BestSubGraphBlocking*) ist eine Erweiterung des *RelatedSubGraphBlockings*, in der die Qualität zwischen Ontologie-Subgraphen beider Ontologien einbezogen wird. Damit wird eine höhere Reduktion des Suchraums ermöglicht (siehe Kapitel 3.3).

- Clusterbasiertes Blocking (*ClusterBlocking*) nutzt ein initiales Mapping, um den Ausgangspunkt für das Clustern zu ermöglichen. In den nächsten Schritten werden die Cluster um den Kontext (Eltern/Kinder) des Konzepts aus dem Ausgangsmapping expandiert (siehe Kapitel 3.4).

- Tokenbasiertes Blocking (*TokenBlocking*) verwendet die Informationen über die Konzeptattribute wie Name und Synonym, die tokenisiert werden. Das heißt, die Attribute werden in Wortkonstrukte zerlegt (beispielweise werden aus dem Namen "*neurologic organ system*" drei Token gebildet: *neurologic*, *organ* und *system*). Die Konzepte beider Ontologien werden anhand der gemeinsamen Token gruppiert (siehe Kapitel 3.4).

- KMeans-Tokenbasiertes Blocking (*KMeansTokenBlocking*) ist ein Ansatz, der Tokenisierung und KMeans-Clustern einsetzt (siehe Kapitel 3.6).

Die Ergebnisse der Arbeit sollen dazu dienen, eine Einschätzung über Effizienz und Effektivität der Blocker zu ermöglichen. Ziel ist es, zu bestimmen, wie sinnvoll die Anwendung verschiedener Blocking-Verfahren für das Matching sehr großer Ontologien aus dem Bereich der Lebenswissenschaften ist.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in 4 Kapitel. Nach einer kurzen Darlegung der Ziele und des Aufbaus der Arbeit im ersten Kapitel werden im zweiten Kapitel die für die Arbeit notwendigen Grundlagen erläutert sowie verwandte Arbeiten diskutiert. Das dritte Kapitel stellt die unterschiedlichen Blocking-Verfahren vor. Das vierte Kapitel umfasst die vergleichende Evaluierung der Blocking-Verfahren. Abschließend wird eine Zusammenfassung und Diskussion der Ergebnisse sowie ein Ausblick für mögliche zukünftige Arbeiten gegeben.

2 Grundlagen

Das Ziel dieses Kapitels ist, die für diese Arbeit relevanten Grundlagen zu Ontologien und Ontologie Mappings vorzustellen. Darüber hinaus werden verwandte Arbeiten zu Blocking-Verfahren diskutiert (Kapitel 2.3).

2.1 Ontologien und Ontologie Mappings

Laut Gruber [11, S. 199] ist eine Ontologie „*a formal, explicit specification of a shared conceptualization*“. Mit anderen Worten ist eine Ontologie ein abstraktes maschinenverstehbares Modell, das die Bedeutungen aller Begriffe definiert und gemeinsam nutzt.

In dieser Arbeit wird folgendes Modell der Ontologie O verwendet: O = (C, R, A). Eine Ontologie O enthält Konzepte C, welche Attribute aus A besitzen und die durch Beziehungen aus R miteinander verbunden sind. Ein Attribut eines Konzepts besitzt einen Namen und einen Wert, z. B. Namen und Synonyme.

Eine Domäne kann von mehreren Ontologien beschrieben werden, die teilweise überlappen können. Aufgrund dessen lässt sich ein Ontologie-Mapping als eine Menge von Korrespondenzen zwischen den Ontologien O_1 und O_2 definieren: $M = \{(c_1, c_2, sim) | c_1 \in O_1, c_2 \in O_2, sim \in [0,1]\}$. c_1 und c_2 stellen die Konzepte aus den Ontologien O_1 und O_2 dar. Die Ähnlichkeit zwischen diesen Konzepten wird als *sim* bezeichnet und liegt im Bereich [0,1]. Je größer der Ähnlichkeitswert ist, desto ähnlicher sind die korrespondierenden Konzepte.

2.2 Ontologie Matching

Für die Identifizierung von semantischen Korrespondenzen zwischen den unterschiedlichen Ontologien kommen semi-automatische Match-Verfahren zum Einsatz [6, 21]. Manuelle Bestimmung ist oft auf Grund der Ontologiegröße nicht möglich. Die Matching-Systeme können mehrere Match-Algorithmen oder Matcher verwenden. In [22]

werden zwei grundlegende Arten von Matchingverfahren unterschieden: Instanz- und Metadatenbasierte Matchingverfahren. Metadatenbasierte Matcher betrachten nur Informationen der Ontologien selbst, d. h. keine Instanzdaten. Die zur Verfügung stehenden Informationen beinhalten konzeptassoziierte Informationen wie Name, Synonym, Beschreibung, und Struktur-Informationen wie "*Parent/Child*"-Beziehungen. Instanzbasierte Matcher untersuchen die Instanzdaten, da sie wichtige Hinweise bezüglich Inhalt und Bedeutung der Ontologiekonzepte geben können.

Typischerweise werden Kombinationen von verschiedenen Matchern eingesetzt sowie unterschiedliche Selektions- und Aggregationsstrategien verwendet.

2.3 Verwendete Ontologien

Für die Auswertung von Blocking-Verfahren sollen im Rahmen dieser Arbeit drei Matchprobleme aus dem Bereich Lebenswissenschaften untersucht werden: FMA-NCIT, FMA-SNOMED und SNOMED-NCIT.

FMA[5]

FMA (*Foundational Model of Anatomy*) ist eine Domain-Ontologie, die das explizite und deklarative Wissen über die menschliche Anatomie modelliert. FMA enthält Klassen, Typen und Beziehungen, die eine symbolische Darstellung der phänotypischen Struktur des menschlichen Körpers in einer maschinenverstehbaren Form ermöglichen. FMA besteht aus vier zusammenhängenden Komponenten:

- Anatomy taxonomy (**At**) klassifiziert anatomische Entitäten, basierend auf den Merkmalen, die sie teilen und durch die sie sich voneinander unterscheiden (siehe Tabelle 2.3-1). Die dominierende Klasse in **At** ist die *Anatomische Struktur*. *Anatomatische Strukturen* enthalten sämtliche Strukturen, die durch reguläre Genexpression entstehen: biologische Makromoleküle, Zellen und ihre

[5] Stand: 08.05.2013 http://sig.biostr.washington.edu/projects/fm/

Komponenten, Gewebe, Organe und Organteile sowie Organsysteme und Körperregionen.

- Anatomical Structural Abstraction (**ASA**) spezifiziert Teil-Ganzes sowie räumliche Beziehungen, die zwischen Entitäten aus **At** entstehen.

- Anatomical Transformation Abstraction (**ATA**) differenziert die morphologische Transformation von Entitäten aus **At** während der pränatalen Entwicklung und des postnatalen Lebenszyklus.

- Metaknowledge (**Mk**) legt die Grundsätze, Regeln und Definitionen fest, nach denen die Klassen und Beziehungen der drei anderen Komponenten der FMA modelliert werden sollen.

Tabelle 2.3-1 Anatomy Taxonomy: Toplevel-Konzepte

Anatomical entity
Attribute entity
Dimensional entity
Spatial association value
Deprecated term

NCIT[6]

Der NCI Thesaurus (NCIT) ist eine Taxonomie, die eine Terminologie im Kontext der Krebsforschung zur Verfügung stellt. NCIT enthält unter anderem Informationen zu Krebskrankheiten, zu Befunden und Auffälligkeiten, zur Anatomie, zu Medikamenten und Chemikalien sowie zu Genen und Genprodukten. Für bestimmte Bereiche, wie Krebserkrankungen und Chemotherapien, stellt es eine detaillierte und einheitliche

[6] http://bioportal.bioontology.org/ontologies/1032.

Terminologie zur Verfügung. Es verknüpft die Terminologie aus der Krebsforschung mit zahlreichen verwandten Bereichen und bietet eine Möglichkeit, diese Arten von Informationen durch semantische Beziehungen zu integrieren oder zu vernetzen. Die NCIT-Terminologie basiert auf dem aktuellen Stand der Wissenschaft und hilft Menschen und Software-Anwendungen, die Ergebnisse der Krebsforschung zu verbinden und zu organisieren [4]. Der Thesaurus enthält derzeit über 100.000 Terme, die in 20 Top Level Kategorien (Is-A Hierarchien) eingeteilt sind (siehe Tabelle 2.3-2).

Tabelle 2.3-2 NCIT: Top Level Kategorien

Abnormal Cell	Experimental Organism Anatomical Concept
Activity	Experimental Organism Diagnosis
Anatomic Structure, System, or Substance	Gene
Biochemical Pathway	Gene Product
Biological Process	Manufactured Object
Chemotherapy Regimen or Agent Combination	Molecular Abnormality
Conceptual Entity	NCI Administrative Concept
Diagnostic or Prognostic Factor	Organism
Disease, Disorder or Finding	Property or Attribute
Drug, Food, Chemical or Biomedical Material	Retired Concept

Die Abbildung 2.3-1 stellt den Anatomie-Teil des NCIT mit den Beziehungen zu den direkten Subkonzepten des Wunschkonzepts dar.

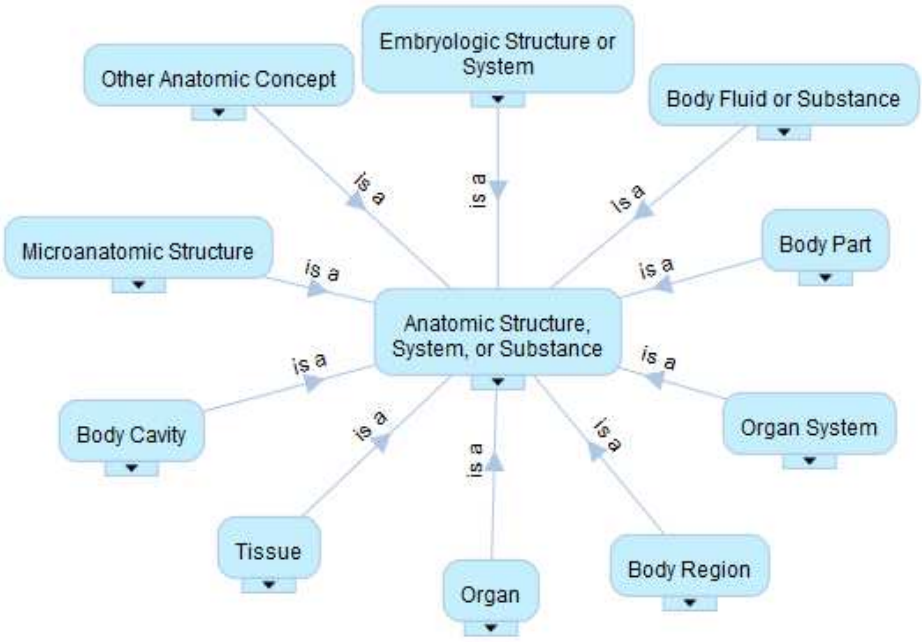


Abbildung 2.3-1 NCIT: Anatomic Structure, System, or Substance.[7]

SNOMED CT[8]

SNOMED CT (*Systematized Nomenclature of Medicine Clinical Terms*) ist eine umfassende und mehrsprachige klinische Terminologie, die zur Verbesserung der Patientenversorgung beitragen soll. Die Terminologie fasst die Konzepte, Terme und Beziehungen zusammen, mit dem Ziel eine umfangreiche Darstellung von klinischen Informationen innerhalb des Gesundheitswesens zu bilden. SNOMED CT kann unter anderem zur Annotation von elektronischen Patientenakten eingesetzt werden, um ein verbessertes Retrieval und eine verbesserte Analyse der Daten zu ermöglichen. Der Inhalt ist in Hierarchien unterteilt, deren Top Level Konzepte in der Tabelle 2.3-3 dargestellt sind.

[7] Stand: 08.05.2013. http://bioportal.bioontology.org/ontologies/50148/?p=terms&conceptid=C12219.
[8] http://www.ihtsdo.org/snomed-ct/.

Tabelle 2.3-3 SNOMED: Top Level Konzepte

Clinical finding/disorder	Physical force
Procedure	Event
Observable entity	Environments and geographic locations
Body Structure	Social context
Organism	Staging and scales
Substance	Linkage concept
Pharmaceutical/biologic product	Qualifier value
Specimen	Special concept
Physical object	Record Artifact

2.4 GOMMA

Für die automatische Erstellung von Mappings zwischen Ontologien und die Verwendung von Blocking-Verfahren soll das GOMMA-Framework verwendet werden. GOMMA steht für *G*eneric *O*ntology *M*atching and Mapping *M*anagement und ist in [15] näher vorgestellt.

GOMMA ist eine generische Infrastruktur zur Verwaltung und Analyse von Ontologien und deren Evolution. Ursprünglich wurde GOMMA für die Untersuchung der Evolution von Ontologie und Mappings in den Lebenswissenschaften entwickelt. Es enthält unter anderem eine Match-Komponente, die spezielle Verfahren für sehr große Ontologien unterstützt. Paralleles Matching [10] ermöglicht, die Laufzeit zu reduzieren. Dabei können die Matchaufgaben partitioniert und parallel ausgeführt werden, oder es erfolgt eine parallele Ausführung des Matchers. Das kompositionsbasierte Verfahren [9] ermöglicht die Wiederverwendung und Komposition der existierenden Mappings, um einen vollständigen

Abgleich zwischen den Ontologien zu vermeiden. Eine einfache Blocking-Methode [8] reduziert den Suchraum.

Die GOMMA-Infrastruktur besteht aus drei Schichten (*repository*, *functional components* und *tools)* und ist in der Abbildung 2.4-1 dargestellt.

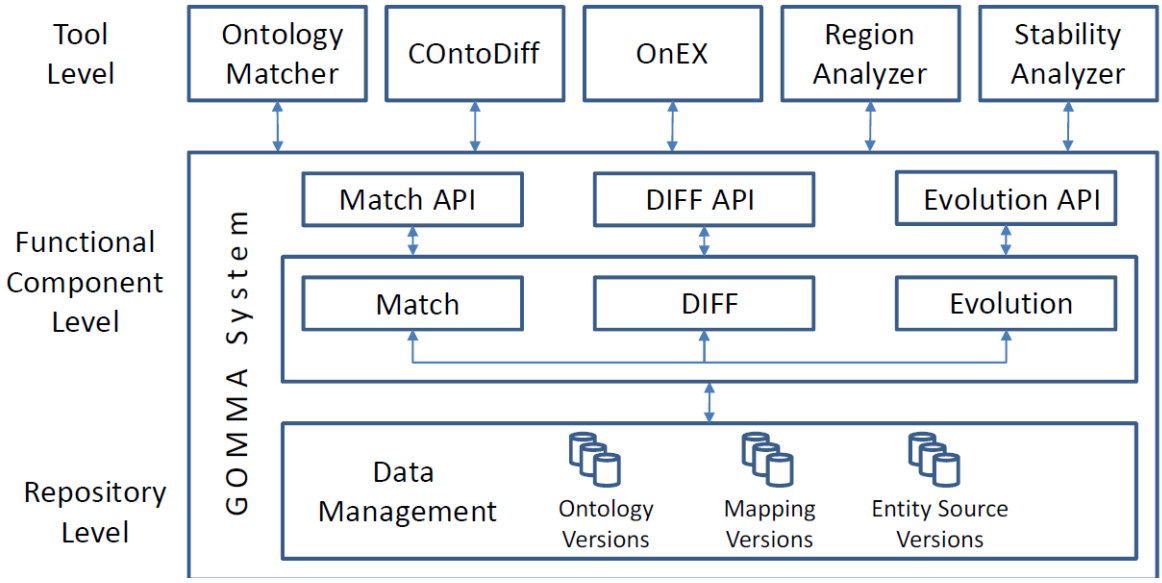


Abbildung 2.4-1 GOMMA Infrastruktur (Quelle: [15, S. 45])

Im Repository werden alle Versionen der Ontologien, deren Entitäten sowie verschiedene Mapping-Arten verwaltet. Die drei funktionalen Komponenten (*functional components*) wie *Match*, *DIFF* und *Evolution* nutzen das Repository für die weitere Verarbeitung.

Die *Match*-Komponente ist dafür verantwortlich, die Mappings zwischen Ontologien zu bestimmen. Die *DIFF*-Komponente dient der Bestimmung von Evolution-Mappings zwischen verschiedenen Versionen einer Ontologie. Es beinhaltet viele Funktionen, um grundlegende Veränderungen wie das Einfügen und Löschen von Konzepten und Beziehungen sowie komplexe Veränderungen wie z. B. das Zusammenführen mehrerer Konzepte in ein Konzept (merge) zu erkennen. Die *Evolution*-Komponente unterstützt die Evolutionsanalyse, in der eine Untersuchung der Historie von Ontologien ermöglicht wird. Diese drei Komponenten haben funktionale Abhängigkeiten. Die *Match*-Komponente

ermittelt Mappings, die später von *DIFF* verwendet werden. Anhand der Ergebnisse der *Diff*-Komponente kann eine Evolutionsanalyse durchgeführt werden.

Für diese Masterarbeit wurde der GOMMA-Match-Workflow verwendet, welcher während der OAEI 2012 [8] eingesetzt wurde. Das Match-Workflow besteht aus drei Kernphasen und ist in der Abbildung 2.4-2 dargestellt.

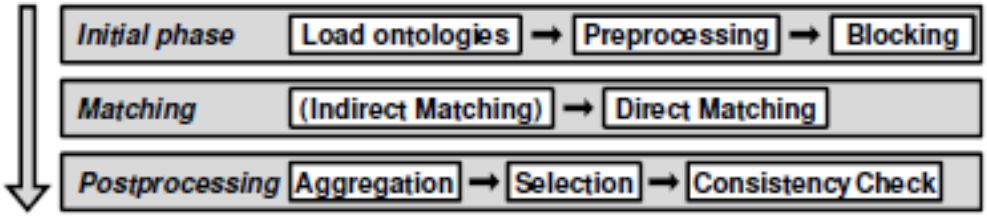


Abbildung 2.4-2 GOMMA OAEI Matching-Workflow. Quelle: [8, S. 2]

In der ersten Phase werden die Ontologien mit allen relevanten Informationen (z. B. Name, Synonyme, Kommentare etc.) in das System geladen. Dazu werden die verfügbaren owl- oder obo-Dateien geparst. Die Konzeptattribute wie z. B. Namen und Synonyme werden normalisiert, d. h. die Trennzeichen und Stopwörter werden entfernt. Zudem wird eine Blocking-Strategie angewendet, um die Anzahl der notwendigen Vergleiche zwischen zweier Ontologien zu reduzieren.

In der Matching-Phase können indirekte und direkte Matching-Strategien verwendet werden. Das indirekte Matching basiert auf der Komposition von existierenden Mappings über eine oder mehrere Mediatorontologien. Beim direkten Abgleich zweier Ontologien wird das interne Wissen (z. B. Konzept- oder Stukturinformationen der Ontologien) angewendet.

Beim Postprocessing werden Aggregierungs- (Maximum, Average) und Selektierungs-Funktionen (*Threshold*, *MaxDelta*) eingesetzt, um die Mappingqualität zu verbessern.

Maximum nimmt nur den maximalen Ähnlichkeitswert, wohingegen ***Average*** den durchschnittlichen Ähnlichkeitswert der Namen und Synonyme zweier Konzepte zurückgibt. Nach der Aggregierung erfolgt die Selektion von Mappings mit besseren Ähnlichkeitswerten mittels *Threshold* und *MaxDelta*.

2.5 Verwandte Arbeiten

Die Berechnung von Korrespondenzen zwischen Ontologien wird durch die Verwendung automatischer Matching-Verfahren [21, 5] ermöglicht. Jedoch sind viele existierende Match-Verfahren und Systeme nicht in der Lage oder haben Probleme, besonders große Ontologien abzugleichen. Daher wurden einige skalierbare Verfahren zum Abgleich sehr großer Ontologien entwickelt. Der Survey in [21] gibt einen Überblick zu existierenden skalierbaren Match-Verfahren, von denen einige im Folgenden diskutiert werden:

Early pruning

Die Idee vom frühen Verwerfen unähnlicher Konzeptpaare (early pruning) [21] besteht darin, die Evaluierung des kartesischen Produktes auf wenige Schritte des Matching-Workflows zu begrenzen und alle Element-Paare mit sehr niedrigen Ähnlichkeitswerten für die weitere Bearbeitung zu eliminieren. Dieser Ansatz ist insbesondere für sequenzielle Matching-Workflows (d. h. die sequenzielle Verarbeitung mehrerer voneinander abhängiger Matcher) geeignet. In [20] wird vorgeschlagen, die Filter-Operatoren wie *Threshold* für diesen Zweck zu verwenden. Das *Threshold* wird entweder statisch vordefiniert oder dynamisch von dem im Matching-Workflow verwendeten Ähnlichkeits-*Threshold* abgeleitet.

Paralleles Matching

Die Arbeit in [10] unterscheidet zwischen Inter- und Intra-Matcher Parallelisierung. Die Inter-Matcher Parallelisierung (siehe Abbildung 2.5-1) ermöglicht die parallele Ausführung von unabhängig voneinander ausführbaren Matchern auf mehreren Prozessoren, um eine schnellere Match-Verarbeitung zu gewährleisten. Die Match-

Ergebnisse können nach unterschiedlichen Aggregierungs- und Selektions-Funktionen kombiniert werden, um das Endergebnis zu erhalten. Die Intra-Matcher Parallelisierung (siehe Abbildung 2.5-2) befasst sich mit der internen Dekomposition der einzelnen Matcher in mehrere Match-Aufgaben, die dann parallel ausgeführt werden können. Dazu muss eine Partitionierung der Eingabeontologien erfolgen. Eine größenbasierte Partitionierung stellt eine einfache Form der Ontologie-Partitionierung dar, in dem beide Ontologien in Partitionen zerlegt werden, die aus einer bestimmten Anzahl von Konzepten bestehen. Die Intra-Matcher Parallelität kann sowohl für sequentiell ausführbare als auch für unabhängige Matcher angewendet und mit der Inter-Matcher Parallelität kombiniert werden.

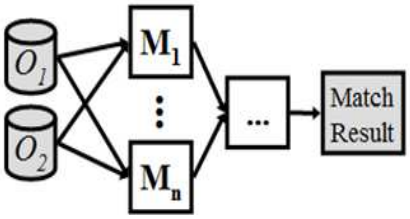


Abbildung 2.5-1 Inter-Matcher Parallelisierung

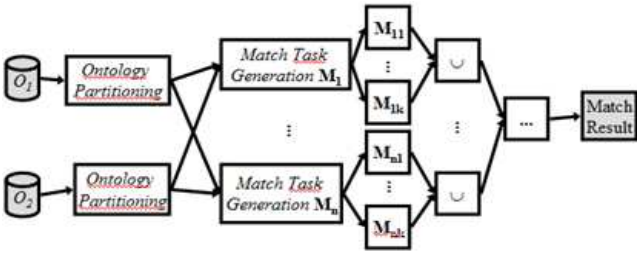


Abbildung 2.5-2 Intra-Matcher Parallelisierung

Wiederverwendung von den existierenden Match-Ergebnissen

Der in [9] beschriebene kompositionsbasierte Match-Algorithmus verwendet existierende Ontologie-Mappings für einen indirekten Abgleich von Ontologien. In Abbildung 2.5-3 haben O_1 und O_2 Mappings zu zwei Mediatorontologien IO_1 und IO_2. Der Ansatz nutzt diese Mappings, um Mappings zwischen O_1 und O_2 auf Basis einer Mapping-Komposition zu bestimmen. Die Wiederverwendung und Komposition der existierenden Mappings vermeidet einen vollständigen, direkten Abgleich von O_1 und O_2.

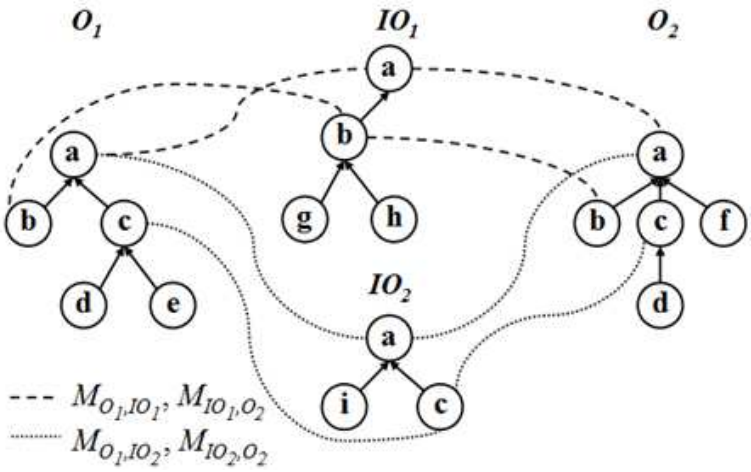


Abbildung 2.5-3 Kompositionsbasiertes Matching

Blocking

Das Ziel von Blocking-Verfahren [1, 8, 12, 13] ist es, den Suchraum während des Matchings zu reduzieren. Bisher existieren einige Partitionierungsstrategien, deren Ziel es ist, Ontologien in Blöcke bzw. Partitionen einzuteilen, um dann während des Matchings nur einen Teil der Blöcke miteinander zu vergleichen. Beispielweise wird im Falcon-System [13] ein "Teile-und-Herrsche"-Ansatz angewendet, dessen Fokus auf Ontologien im OWL- oder RDF-Format liegt (siehe Abbildung 2.5-4).

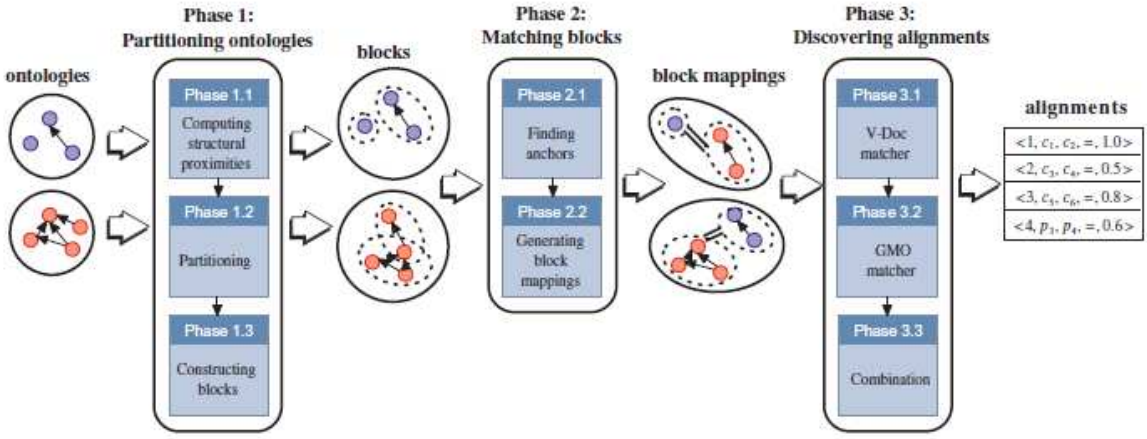


Abbildung 2.5-4 Der "Teile-und-Herrsche"-Ansatz. Quelle[13, S.141]

Der "Teile-und-Herrsche"-Ansatz ist in drei Phasen unterteilt: *Partitioning ontologies*, *Matching blocks* und *Discovering alignments*. Die Ontologie-Entitäten (z. B. Klassen und deren Eigenschaften) werden auf Basis ihrer strukturellen Ähnlichkeiten in eine Menge kleiner Cluster zerlegt. Nach der Berechnung der strukturellen Ähnlichkeiten werden alle Ontologieentitäten in eine Menge disjunkter Cluster partitioniert. Die gebildeten Blöcke werden mit Hilfe von vorberechneten Ankern (anchors) gematcht. Die Ähnlichkeitsberechnung zwischen zwei Ontologiekonzepten erfolgt nicht nur unter Berücksichtigung der Gemeinsamkeiten zwischen Beschreibungen der Entitäten, sondern auch ihrer Unterschiede. Falls die Ähnlichkeit größer als ein vordefinierter Wert ist, werden die Ontologiekonzepte mit diesen Beschreibungen als Anker betrachtet. Falsche Anker werden manuell entfernt. Die Ähnlichkeit zwischen den Blöcken kann auf Basis der Anker-Verteilung berechnet werden. Die Grundidee besteht darin, dass die Blöcke umso ähnlicher sind, je mehr Anker zwischen zwei Blöcken gefunden werden. Jeder Block in einem Block-Mapping ist eine kleine Sub-Ontologie. Verschiedene Matching-Ansätze können verwendet werden, um die Alignments zwischen Blöcken in jedem Block-Mapping zu bestimmen.

Ziel des clusterbasierten Ansatzes für sehr große Ontologien [1, S. 5] ist die Partitionierung des Schema-Graphs (SG) in eine Menge disjunkter Subgraphen, um den Abgleich zwischen gesamten Schema-Graphen der Ontologien zu erleichtern. Der Schema-Graph ist die Repräsentation einer Ontologie als gerichteter azyklischer Graph, dessen Knoten und

Kanten jeweils die Konzepte und Beziehungen zwischen den Konzepten darstellen. Der Algorithmus basiert auf den Kontextähnlichkeiten zwischen den Knoten, so dass strukturell ähnliche Knoten zu einem Cluster zugeordnet werden, während die Knoten von unterschiedlichen Clustern strukturell unähnlich sind. Unter Kontext sind alle Vorfahren, Nachkommen und Geschwister des Knotens gemeint. Der Algorithmus erzeugt den Baum, welcher eine "bottom-up"-Hierarchie von Clustern (Dendrogramm) repräsentiert. Zuerst stellt jeder Knoten einen Cluster dar, der nur aus diesem Knoten besteht. Danach werden die Knoten des SG iterativ anhand deren Strukturähnlichkeit gemergt, um eine Hierarchie zu bilden. Jede Ebene im Dendrogramm ist mit einem Wert verbunden, der den Durchschnittswert der Intra-Cluster-Ähnlichkeiten der Cluster auf dieser Ebene darstellt. Der Intra-Cluster-Ähnlichkeitswert zeigt, wie ähnlich die Knoten innerhalb eines Clusters sind. Für diesen Zweck werden die Ähnlichkeitswerte zwischen jedem Paar innerhalb des Clusters berechnet und ein Mittelwert unter Berücksichtigung der Anzahl der Knoten ermittelt. Zum Schluss wird die Cluster-Menge einer Ebene ausgewählt, die den besten Intra-Cluster-Ähnlichkeitswert aufweist.

Das Ziel des aligment-basierten Ansatzes [12] ist die Partitionierung der Ontologien in eine Menge von Blöcken, so dass nur Blöcke von angemessener Größe verarbeitet werden. Aus zwei entstehenden Blockmengen werden die Paare gebildet, die weiter verglichen werden. Dafür werden zwei Methoden vorgeschlagen: *PAP (Partition, Anchor, Partition)* und *APP (Anchor, Partition, Partition)*. Die Auswahl des Verfahrens hängt vom Strukturierungsgrad der Ontologien ab: Ist eine Ontologie (O_T) strukturierter als die andere (O_S), wird *PAP* verwendet. Haben beide Ontologien einen hohen Strukturierungsgrad, wird *APP* eingesetzt. Die Idee des PAP besteht darin, dass die höher strukturierte Ontologie mit Hilfe des PBM-Algorithmus [12. S. 4] in mehrere Blöcke (B_{Ti}) partitioniert wird. Danach werden die Zentren (CB_{Si}) der zukünftigen Blöcke O_S identifiziert, indem zwei Kriterien berücksichtigt werden: die vorberechneten Ankerpaare zwischen den Ontologien O_T und O_S und die gebildeten Blöcke B_{Ti}.. Als Anker dienen die Konzepte, die in beiden Ontologien gleiche Labels haben. Im nächsten Schritt wird die Ontologie O_S rund um diese Zentren in Blöcke (B_{Si}) zerlegt. Zum Schluss bildet jede Block B_{Si} mit dem entsprechenden Block B_{Ti} (anhand CB_{Si}) ein Paar. Bei der *APP*-Methode werden beide Ontologien mit dem

modifizierten PBM-Algorithmus in mehrere Blöcke partitioniert. Danach werden die Blockpaare anhand der Anzahl der Anker, die sie teilen, gebildet.

Im Rahmen dieser Arbeit werden Blocking-Verfahren entwickelt bzw. umgesetzt, die semantische sowie strukturale Aspekte der Ontologie betrachten.

3 Blocking Verfahren

In diesem Kapitel werden verschiedene Blocking-Ansätze vorgestellt und diskutiert. Diese werden innerhalb des in Abbildung 2.4-2 vorgestellten Matching-Workflows eingesetzt. Dafür soll eine einheitliche Schnittstelle für alle Blocker in GOMMA zur Verfügung gestellt werden.

Jeder Blocker erhält zwei Ontologien (O_1 und O_2) als Eingabe. Ausschließend wird ein Algorithmus zur Partitionierung der Eingabe-Ontologien angewendet (siehe Algorithmus 1). Jeder Blocking-Algorithmus erstellt mehrere sogenannte *PartitionPairs* (bzw. ein *PartitionPair*). Ein *PartitionPair* besteht aus zwei Ontologie-Teilen (*DomainPartition* und *RangePartition*), die jeweils Konzepte aus O_1 (domain-Ontologie) bzw. O_2 (range-Ontologie) enthalten. Die Domain- und Range-Partition eines *PartitionPairs* werden anschließend in der Match-Phase abgeglichen. Somit werden nur Teile der Ontologien verglichen, mit dem Ziel, die Evaluierung des gesamten kartesischen Produkts bezüglich der Ontologiegröße zu vermeiden. Die Ergebnisse des Matchings der einzelnen *PartitionPairs* werden einschließend zu einem Mapping vereinigt und ausgegeben.

Algorithm *Blocking* (O_1, O_2)

Input: Two ontologies O_1 and O_2
Output: *PartitionPair*

1: *domainPartition* ← *BuildPartition*(O_1)
2: *rangePartition* ← *BuildPartition*(O_2)
3: *PartitionPair* ← *buildPartitionPair(domainPartition, rangePartition)*
4: **return** *PartitionPair*

Algorithmus 1 Blocking

3.1 Blocking von Ontologie-Subgraphen

Das Blocking von Ontologie-Subgraphen [8] (siehe Algorithmus 2) basiert auf dem initialen Mapping (*IM*). *IM* wird mit Hilfe eines effizienten Match-Verfahrens berechnet (z. B. exaktes Matching auf indexierten Konzeptnamen) (Zeile 3).

Algorithm *OntoSubBlocking*(O_1, O_2)

Input: Two ontologies O_1 and O_2
Output: *PartitionPair*

1: *domainPartition* ← {}
2: *rangePartition* ← {}
3: *IM* ← *computeIM();*
4: *domainPartition* ← *BuildPartition*(O_1, *IM.getDomainConcepts()*)
5: *rangePartition* ← *BuildPartition*(O_2, *IM.getRangeConcepts()*)
6: *PartitionPair* ← *buildPartitionPair(domainPartition, rangePartition)*
7: **return** *PartitionPair*

Algorithmus 2 OntoSubBlocking

Für die Partitionsbildung (Zeilen 4-5) ist es nötig die Subgraphwurzeln zu bestimmen. Da die Ontologien einen gerichteten Graphen darstellen, bestehen sie aus mehreren Knoten, die in eine Hierarchie eingeordnet sind. Die Subgraphwurzeln sind die Wurzeln des Teilgraphen, die der Ontologie-Graph beinhaltet. In Abbildung 3.1-1 sind b_1, e_1, h_1, etc. solche Wurzeln. Die Subgraphwurzeln werden im zweiten Schritt identifiziert (Algorithmus 3, Zeile 5), und die Anzahl von Korrespondenzen wird aus dem Ausgangsmapping pro Subgraphwurzel bestimmt (Algorithmus 3, Zeilen 7-12). Die Kernidee besteht darin, dass zuvor bestimmte Korrespondenzen des initialen Mappings von den Blättern nach oben propagiert werden. Im Fall von Mehrfachvererbung werden die Korrespondenzanzahlen auf Elternknoten gleichmäßig aufgeteilt (Algorithmus 3, Zeile 9). Abbildung 3.1-1 zeigt ein Beispiel zur Veranschaulichung dieses Prozesses. Beispielweise hat c_2 in O_2 zwei Elternkonzepte p_2 und r_2, d. h. jedes Elternkonzept bekommt die Korrespondenz-Anzahl von 0,5. Im nächsten Schritt wird für jede Wurzel der Anteil der Korrespondenzen des Subgraphs der Wurzel im Vergleich zur Gesamtanzahl an Korrespondenzen des initialen Mappings (*corrFrac*) bestimmt. (Algorithmus 3, Zeile 16).

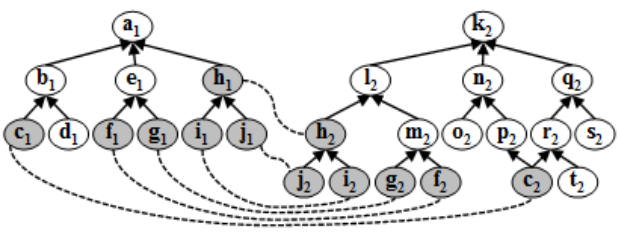
Algorithm *BuildPartition* (*O, IMConcepts*)

Input: Ontology *O*, concepts from O, covered by initial Mapping *IMConcepts*
Output: *partition* - concepts from O

```
1:  roots ← {}
2:  rootCorrCount ← {}
3:  partition ← {}
4:  for each concept ∈ IMConcepts do
5:     roots ← getRootForConcept(O, concept)
6:     for each root ∈ roots do
7:        if rootCorrCount.contain(root) do
8:           currentCount ← rootCorrCount.getCount(root)
9:           rootCorrCount.put(root, (currentCount + 1/roots.size()))
10:       else
11:          rootCorrCount.put(root, (1/roots.size()))
12:       end if
13:    end for
14: end for
15: for each rootConcept ∈ rootCorrCount do
16:    corrFrac ← rootCorrCount .get(rootConcept) /IM.size()
17:    if corrFrac > threshold do
18:       subGraph ← getSubGraph(O, rootConcept))
19:       partition addObjects(subGraph)
20:    end if
21: end for
22: return partition
```

Algorithmus 3 BuildPartition

Anschließend werden nur die Wurzeln ausgewählt, deren *corrFrac* über einem gegebenen *Threshold* liegt (Algorithmus 3, Zeilen 17-20). Alle Konzepte, die im Subgraph der selektierten Wurzeln liegen, werden für das weitere Matching verwendet. Falls keine Wurzeln vorhanden sind, die über dem gegebenen *Threshold* liegen, wird die ganze Ontologie abgeglichen. Dies ist ein defensiver Ansatz, da Teile der Ontologie (d. h. einige Subgraphen) nur verworfen (geblocked) werden, wenn mindestens eine Wurzel besonders viele Korrespondenzen unter sich versammelt (siehe Abbildung 3.1-1).



$$corrFrac(root) = \frac{|M_{initial}(subgraph(root))|}{|M_{initial}|}$$

$$corrFrac(b_1) = \frac{1}{6} = 0.16 \quad corrFrac(l_1) = \frac{5}{6} = 0.83$$

$$corrFrac(e_1) = \frac{2}{6} = 0.33 \quad corrFrac(n_2) = \frac{0.5}{6} = 0.083$$

$$corrFrac(h_1) = \frac{3}{6} = 0.5 \quad corrFrac(q_2) = \frac{0.5}{6} = 0.083$$

Abbildung 3.1-1 Blocking von Ontologie-Subgraphen. Quelle: [8, S. 3]

3.2 Blocking anhand zusammenhängender Ontologie-Subgraphen

Das Ziel des Blockings anhand zusammenhängender Ontologie-Subgraphen ist es, alle Subgraphen der Ontologien O_1 und O_2 zu finden, die miteinander durch Korrespondenzen verknüpft sind. Algorithmus 4 illustriert den *RelatedSubGraphBlocking*-Ansatz. Wie beim *OntoSubBlocking* basiert das Verfahren auf einem initialen Mapping (*IM*, Zeile 2), das mit Hilfe eines effizienten Match-Verfahrens berechnet wird.

Algorithm *RelatedSubGraphBlocking(O_1, O_2)*

Input: Two ontologies O_1 and O_2
Output: *PartitionPair*- Partition Pair

```
1:  PartitionPairSet ← {}
2:  IM ← computeIM();
3:  rootPairs ← findRootPairs (O1, O2, IM )
4:  domainPartition ← {}
5:  rangePartition ← {}
6:  for each rootpair∈ rootPairs do
7:      subGraphD ← getSubGraph(O1, rootpair.getDomain()))
8:     domainPartition addObjects(subGraphD)
9:     subGraphR ← getSubGraph(O1, rootpair.getRange()))
10:    rangePartition addObjects(subGraphR)
11: end for
12: PartitionPair ← buildPartitionPair(domainPartition, rangePartition)
13: return PartitionPair
```

Algorithmus 4 Blocking anhand zusammenhängender Ontologie-Subgraphen

Um die Wurzelpaare zu finden (Algorithmus 4, Zeile 3), werden für jede Korrespondenz aus *IM* die Subgraphwurzeln der beiden Ontologien identifiziert (Algorithmus 5, Zeilen 3-4). Somit bilden die zusammenhängenden Subgraphwurzeln die Wurzelpaare (*rootPairs*). In der Abbildung 3.2-1 sind beispielhaft die Subgraphen der Ontologien O_1 und O_2 dargestellt. Nach der Anwendung des zuvor beschriebenen Algorithmus werden folgende Wurzelpaare gebildet: a_1 - d_2; a_1 - f_2, b_1 - d_2 und b_1 - e_2. Da keine Verknüpfung im Subgraphen von p_1 zu Konzepten in O_2 liegt, wird p_1 keinem Paar zugewiesen.

Algorithm *findRootPairs* (O_1, O_2, *IM*)

Input: Two ontologies O_1 and O_2, InitialMapping *IM*
Output: *rootPairs* - set of rootPairs

```
1:  rootPairs ← {}
2:  for each corr ∈ IM do
3:      rootDomain ← getRootForConcept(O1, corr.getDomainConcept)
4:      rootRange ← getRootForConcept(O2, corr.getRangeConcept)
5:      rootPair ← buildRootPair(rootDomain , rootRange )
6:      if !rootPairs.contains(rootPair) do
7:          rootPairs .add(rootPair)
8:      end if
9:  end for
10: return rootPairs
```

Algorithmus 5 FindRootPairs

Im letzten Schritt werden für jedes Wurzelpaar alle Konzepte, die in deren Subgraphen liegen, zur Domain- bzw. Range-Partition hinzugefügt (Algorithmus 4, Zeilen 6-11), so dass in diesem Blocking-Algorithmus ein *PartitionPair* (Zeile 12) erzeugt wird. Dies *PartitionPair* wird im anschließenden Matching verwendet.

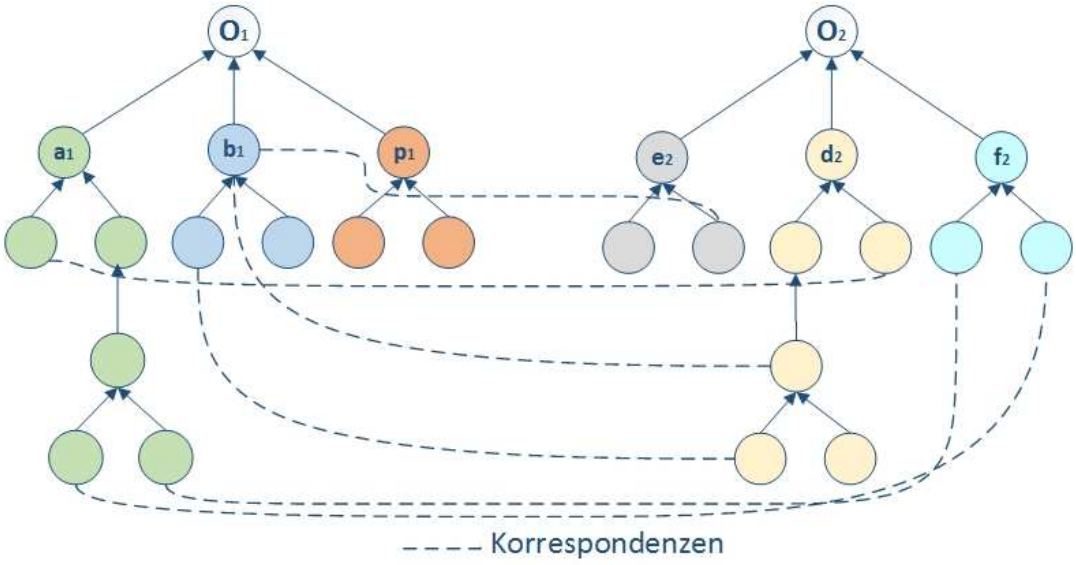


Abbildung 3.2-1 Subgraphen der Ontologien

3.3 Blocking anhand der am besten verknüpften Ontologie-Subgraphen

Um ein Blocking anhand der am besten verknüpften Ontologie-Subgraphen zu ermöglichen, werden nur die Subgraphen der Ontologien O_1 und O_2 für das spätere Matching ausgewählt, die die meisten Verknüpfungen zueinander haben (siehe Algorithmus 6). Die Verknüpfungen stellen die Korrespondenzen des initialen Mappings (IM) dar, das mit Hilfe eines effizienten Match-Verfahrens berechnet wird (Algorithmus 6, Zeile 2). Im Unterschied zu der in Kapitel 3.2 vorgestellten Methode wird hier die Qualität der Wurzelpaare (Algorithmus 6, Zeile 3) einbezogen. Wenn es mehrere Subgraphen der Ontologien O_1 und O_2 gibt, die die gleiche maximale Anzahl an Verknüpfungen zueinander haben, können mehrere *PartitionPairs* gebildet werden.

Algorithm *BestSubGraphBlocking(O_1, O_2)*

Input: Two ontologies O_1 and O_2
Output: *PartitionPairSet*- set of Partition Pairs

```
1:  PartitionPairSet ← {}
2:  IM ← computeIM();
3:  rootPairs ← findBestRootPairs (O1, O2, IM )
4:  for each rootpair∈ rootPairs do
5:      domainPartition ←{}
6:      rangePartition ←{}
7:      subGraphD ← getSubGraph(O1, rootpair.getDomain()))
8:      domainPartition.addObjects(subGraphD)
9:      subGraphR ← getSubGraph(O1, rootpair.getRange()))
10:     rangePartition.addObjects(subGraphR)
11:     PartitionPair ← buildPartitionPair(domainPartition, rangePartition)
12:     PartitionPairSet.add (PartitionPair)
13: end for
14: return PartitionPairSet
```

Algorithmus 6 Blocking anhand der am besten verknüpften Ontologie-Subgraphen

Um dies zu ermöglichen, werden zuerst für jede Korrespondenz aus *IM* die Subgraphwurzeln identifiziert (Algorithmus 7, Zeilen 4-5). Die zusammenhängenden Subgraphwurzeln bilden die Wurzelpaare (*rootPairs*). Im nächsten Schritt wird für jedes Wurzelpaar der Anteil der Korrespondenzen bestimmt, d. h. wie viele Korrespondenzen in dem Subgraphen der Wurzel aus dem initialen Mapping liegen (Algorithmus 7, Zeilen 7-8). Die Wurzelpaare, die den größten Anteil an Korrespondenzen haben, werden für die weitere Verarbeitung ausgewählt (Algorithmus 7, Zeilen 10-18). Im Beispiel in Abbildung 3.2-1 sind die Korrespondenzen so verteilt, dass die Wurzelnpaare a_1 - f_2 und b_1 - d_2 den maximalen Anteil an Korrespondenzen haben (je 2 Korrespondenzen aus *IM*).

Algorithm *findBestRootPairs* (O_1, O_2, *IM*)

Input: Two ontologies O_1 and O_2, InitialMapping *IM*
Output: *rootPairs*- list of rootPairs with amount of correspondences between roots

```
1:  rootPairsCount ← {}
2:  for each corr ∈ IM do
3:      rootDomain ← getRootForConcept(O1, corr.getDomainConcept)
4:      rootDomain ← getRootForConcept(O2, corr.getRangeConcept)
5:      rootPair ← buildRootPair(rootDomain , rootRange)
6:      currentCount ← rootPairsCount.getCount(rootPair)
7:      rootPairsCount.put(rootPair, (currentCount + 1))
8:  end for
9:  heightCount ← 0
10: for each corrCount ∈ rootPairsCount do
11:     if corrCount > heightCount do
12:         rootPairs.moveAll()
13:         rootPairs.addAll (rootPairsCount.get(corrCount))
14:         heightCount = corrCount
15:     else if corrCount == heightCount do
16:         rootPairs.addAll (rootPairsCount.get(corrCount))
17:     end if
18: end for
19: return rootPairs
```

Algorithmus 7 FindBestRootPairs

Alle Konzepte, die in den Subgraphen der selektierten Wurzelpaare liegen, bilden die *PartitionPairs.* Die entstehenden *PartitionPairs* werden für das weitere Matching verwendet. Dieser Algorithmus bringt insbesondere dann Vorteile, wenn die Korrespondenzen in einem Subgraphen konzentriert und nicht über mehrere Subgraphen verteilt sind.

3.4 Clusterbasiertes Blocking

Im Unterschied zu den in den Kapiteln 3.1-3.3 vorgestellten Verfahren werden beim clusterbasierten Blocking nicht ganze Subgraphen ausgewählt, sondern relativ kleinere Cluster anhand von Eltern- und Kindbeziehungen und zuvor berechneten Korrespondenzen bestimmt. Die Idee des clusterbasierten Ansatzes besteht darin, dass die Korrespondenzpaare des initialen Mappings (*corr* ∈ *IM*) Cluster bilden (Algorithmus 8).

```
Algorithm ClusterBlocker(O1, O2 )
Input:  Two ontologies O1 and O2,
Output: PartitionPairSet
1:    PartitionPairSet ← {}
2:    IM ← computeIM();
3:    for each corr ∈ IM do
4:      cD← getDomainConcept(corr)
5:      cR ← getRangeConcept(corr)
6:      //Domain
7:      domainCluster ← findContext (cD, IM)
8:      //Range
9:      rangeCluster ←findContext (cR, IM)
10:     if domainCluster >0 && rangeCluster >0 then
11:           PartitionPair ← build (domainCluster, rangeCluster)
12:           PartitionPairSet.add(PartitionPair)
13:     end if
14:   end for
15:   return PartitionPairSet
```

```
Algorithm   findContext (concept, IM)
Input:  concept, initial Mapping IM
Output: cluster
1:  ProcessedConcepts← {}
2:  if ! ProcessedConcepts.contains(concept) then
3:      ProcessedConcepts.add(concept)
4:      cluster.add(concept)
5:      for each predecessor ∈ concept do
6:        if ! ProcessedConcepts.contains(predecessor) then
7:          cluster.add(predecessor)
8:          if IM.contains(predecessor) then
9:           findContext (predecessor, ProcessedConcepts, IM)
10:         end if
11:       end if
12:     end for
13:     for each successor ∈ concept do
14:       if ! ProcessedConcepts.contains(successor) then
15:         cluster.add(successor)
16:         if IM.contains(successor) then
17:           findCotext (successor ProcessedConcepts, IM)
18:         end if
19:       end if
20:     end for
21:  end if
22:return cluster
```

Algorithmus 8 ClusterBlocking

Die Erstellung der Cluster erfolgt laut Algorithmus 8. Ähnlich zu den vorherigen Blocking-Verfahren (siehe Kapitel 3.1) wird im ersten Schritt ein initiales Mapping (*IM*) mit Hilfe eines effizienten Match-Verfahrens berechnet (Algorithmus 8, Zeile 4). Anschließend wird der Konzeptkontext (Eltern, Kinder) der Korrespondenzen des Ausgangs-Mappings bestimmt. Falls die Eltern/Kinder-Konten auch in *IM* enthalten sind und noch keinem Cluster zugewiesen wurden, werden sie sowie ihre Eltern/Kinder zum Cluster hinzugefügt. Solange der Konzeptkontext an einer Korrespondenz in *IM* beteiligt ist, wird dieser Schritt wiederholt. In Abbildung 3.4-1 ist das Konzept k_1 im *IM*, es hat

keine Kinder-, jedoch 2 Eltern-Knoten: a_1 und b_1. Das Konzept b_1 ist auch im *IM* enthalten, d. h. dessen Kontext (w_1 und j_1) wird auch dem Cluster hinzugefügt. Darüber hinaus besteht das entstehende Cluster aus fünf Konzepten. Die erstellten Cluster bilden die *PartitionPairs*. Die *PartitionPairs* werden im anschließenden Matching abgeglichen.

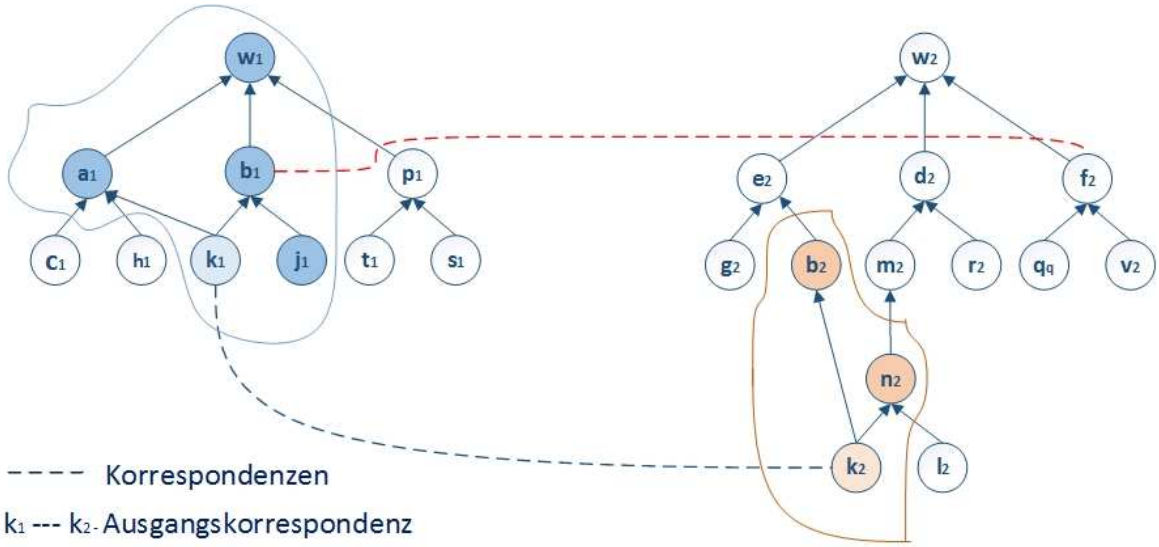


Abbildung 3.4-1 Clusterbasiertes Blocking

Clusterbasiertes Blocking: Erweiterungen

Der oben dargestellte Ansatz ist ein allgemeiner Ansatz, der mit verschiedenen Einstellungen realisiert werden kann:

- **Distanz** – stellt die Entfernung der Eltern/Kinder-Knoten des aktuell betrachteten Konzepts dar. In Abbildung 3.4-1 wurde eine Distanz von 1 für die Bestimmung des Konzepts genutzt. Abbildung 3.4-2 zeigt ein Beispiel, falls die Distanz auf 2 gesetzt wird. Um den Kontext von k_2 in der Ontologie O_2 zu finden, werden neben n_2 und b_2 zusätzlich die Konzepte e_2 , m_2 und l_2 geholt und somit dem Cluster *c3* hinzugefügt. Wenn e_2, m_2 und l_2 ebenfalls an einer Korrespondenz aus *IM* beteiligt

wären, würden sie zusätzlich mit deren zugehörigem Kontext (auch mit der Distanz 2) dem Cluster hinzugefügt.

- **Zugehörigkeit eines Konzepts zu mehreren Clustern**: Im allgemeinen Ansatz wird ein Konzept nur einem Cluster zugeordnet. In manchen Situationen könnte die Zuordnung eines Konzepts zu mehreren Clustern sinnvoll sein. Beispielweise könnte das Konzept w_1 in Abbildung 3.4-2 sowohl dem Cluster c1 als auch dem Cluster *c2* zugewiesen werden.

- **Verbindung des Clusters einer Ontologie mit mehreren Clustern einer anderen Ontologie**: Ein Cluster kann in Verbindung zu mehreren anderen stehen. Beispielweise steht das Cluster *c1* durch k_1--k_2 und b_1--b_2 in einem Zusammenhang zu den Clustern *c3* und *c4* (siehe Abbildung 3.4-2). Um alle Korrespondenzen des initialen Mappings *IM* abzudecken und möglichst viele Mappings mit hoher Qualität zu ermitteln, wird für jedes Paar von verbundenen Clustern in O_1 und O_2 ein *PartitionPair* erzeugt. Dies könnte jedoch zu einer sehr hohen Anzahl an notwendigen Vergleichen führen, wodurch sich die Ausführungszeit erhöhen würde.

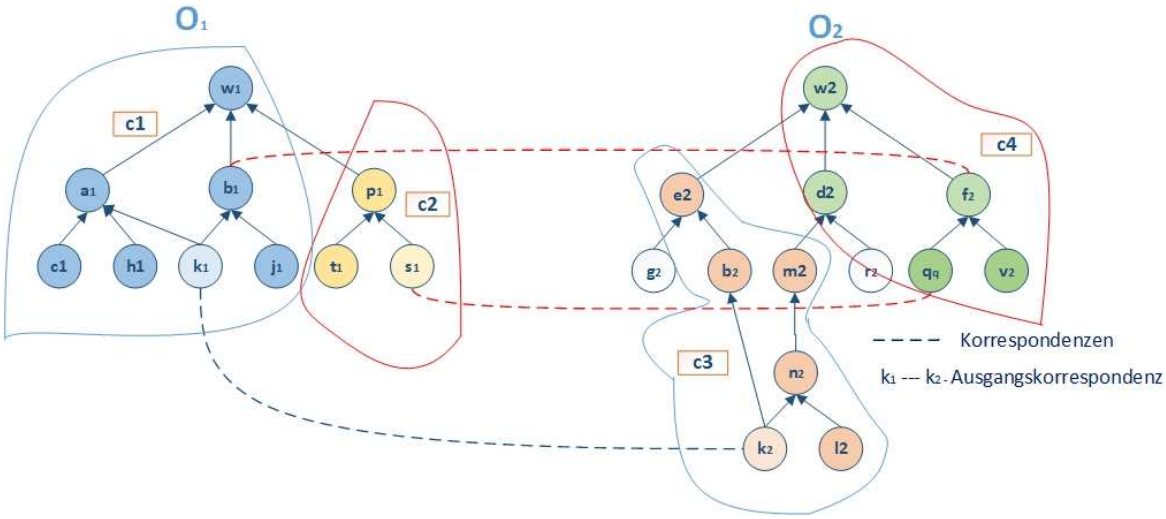


Abbildung 3.4-2 Clusterbasiertes Blocking (Distanz = 2)

3.5 Tokenbasiertes Blocking

Bisher wurden 4 Methoden vorgestellt, die anhand eines effizient bestimmten, initialen Mappings Partitionen der Eingabeontologien bestimmen, die im Matching abgeglichen werden sollen. Dabei wurden miteinander verknüpfte Ontologie-Subgraphen bestimmt. Im Gegensatz dazu sollen im tokenbasierten Blocking Partitionen von Ontologien anhand ähnlicher Tokens der Konzeptnamen bzw. Synonyme erstellt werden. Unter Token sind die Wortkonstrukte gemeint, aus denen die Konzeptattribute bestehen. Beispielweise wird das Attribut "*neurologic organ system*" in drei Token zerlegt: *neurologic*, *organ* und *system*. Der tokenbasierte Ansatz ist in Algorithmus 9 dargestellt.

Algorithm *TokenBlocking*(O_1, O_2,)

Input: Two ontologies O_1 and O_2,
Output: *PartitionPair*

```
1:  ProcessedDConcepts ← {}
2:  ProcessedRConcepts ← {}
3:  domainPartition ← {}
4:  rangePartition ← {}
5:  //Domain
6:  mapsD ←buildTokenMaps(O1 , tokenMapD, tokenConceptsMapD )
7:  tokenMapD ← mapsD.getTokenMap()
8:  tokenConceptsMapD← mapsD.getTokenConceptsMap()
9:  //Range
10: mapsR ← buildTokenMaps(O2 , tokenMapR, tokenConceptsMapR )
11: tokenMapR ← mapsR.getTokenMap()
12: tokenConceptsMapR ← mapsR.getTokenConceptsMap()
13: for each token ∈ tokenMapsD do
14:       if tokenMapD.contains(token) && tokenMapR.contains(token) then
15:           for each cD ∈ tokenConceptsMapD.getConcepts(token) do
16:               for each cR ∈ tokenConceptsMapR.getConcepts(token) do
17:                 if !ProcessedDConcepts.contains(cD) then
18:                   domainPartition.add(cD)
19:                   ProcessedDConcepts.add(cD)
20:                 end if
21:                 if !ProcessedRConcepts.contains(cR) then
22:                   rangePartition.add(cR)
23:                  ProcessedRConcepts.add(cR)
24:                end if
25:              end for
26:           end for
27:       end if
28:  end for
29: PartitionPair ← buildPartitionPair(domainPartition, rangePartition)
30: return PartitionPair
```

Algorithmus 9 TokenBlocking

Um die Gruppierung der Ontologie-Konzepte anhand der geteilten Token zu zu ermöglichen, müssen zwei *TokenMaps* pro Ontologie erstellt werden. Ein *TokenMap*

enthält die Token und deren Vorkommen-Anzahl (<T, F>). Ein weiteres *TokenMap* enthält für jedes Token die Liste der Konzepte, in welchen das Token vorkommt (<T, <C>>) (Algorithmus 9, Zeilen 6-12).

Algorithm *buildTokenMaps*(O)

Input: Ontology O
Output: *maps* - consists of two maps:<token, frequency> and <token,<concepts>>

```
1:   tokenMap ← {}
2:   tokenConceptsMap ← {}
3:   for each concept ∈ O do
4:     tokenList ← tokenize(concept)
5:     for each token ∈ tokenList do
6:       if tokenMap.contains(token) then
7:         frequency← tokenMap.get(token)
8:         tokenMap.put(token, frequency+1)
9:         tokenConceptsMap.getConceptList(token).add(concept)
10:      else
11:        tokenMap.add(token, 1)
12:        tokenConceptsMap.add(token, concept)
13:      end if
14:    end for
15:  end for
16:  maps.add(tokenMap)
17:  maps.add(tokenConceptsMap)
     return maps
```

Algorithmus 10 BuildTokenMaps

Die Bildung des TokenMaps stellt der Algorithmus 10 dar. Zunächst werden für jedes Konzept aus O_1 und O_2 die Attribute wie z. B. der Name und Synonyme extrahiert und tokenisiert (Algorithmus 10, Zeile 4). Die Abbildung 3.5-1 zeigt ein Beispiel, wie die Tokenisierung des NCIT-Konzepts C12755 erfolgt. Danach werden <T, F>- und <T, <C>>-Maps gebildet (Algorithmus 10, Zeilen 5-14), so dass bei jedem neuen Auftreten des Tokens die Anzahl der Vorkommen erhöht wird (Algorithmus 10, Zeilen 7-8) und die neuen Konzepte zur Konzeptliste hinzugefügt werden (Algorithmus 10, Zeile 9).

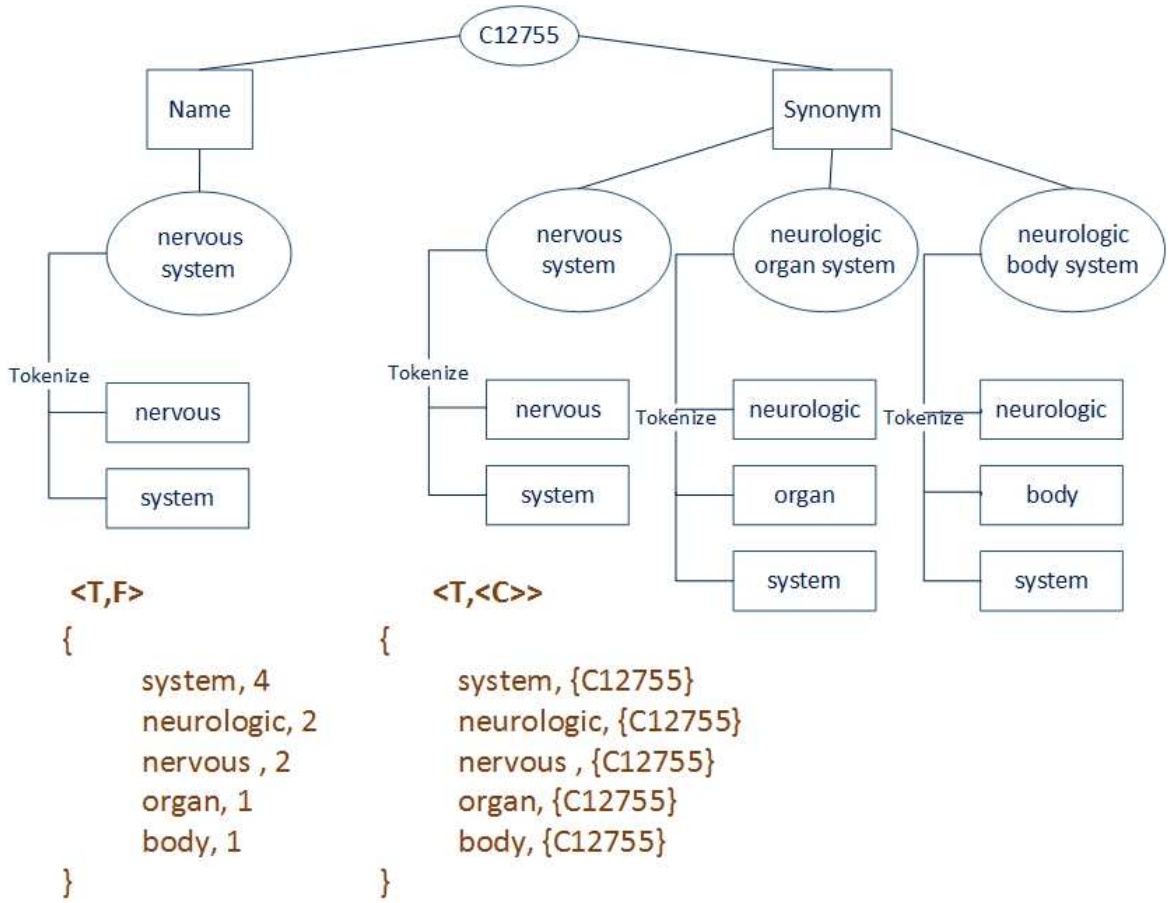


Abbildung 3.5-1 NCIT: Tokenisierung des Konzepts C12755

Danach werden die <*T*, *F*>-Maps sortiert. 2,5% aller Token, die am häufigsten vorkommen, werden aus <*T*, *F*> entfernt. Dies wird durchgeführt, um die nicht aussagekräftigen Token zu eliminieren. In Abbildung 3.5-1 ist <*T*, *F*> in absteigender Reihenfolge sortiert. Das Token "System" kommt am häufigsten vor (4-mal). Anschließend werden nur die Token ausgewählt, die in beiden Ontologien vorkommen. Alle Konzepte aus <*T*, <*C*>>, die die selektierten Token enthalten, werden zur *Domain*- bzw. *Range-Partition* hinzugefügt. Somit wird ein *PartitionPair* gebildet.

Eine weitere Möglichkeit ist, mehrere *PartitionPairs* zu erzeugen (siehe Algorithmus 11). In diesem Fall wird für jedes Token, das in beiden Ontologien vorkommt, ein *PartitionPair* gebildet (siehe Algorithmus 11, Zeilen 17-28). Ein Konzept kann dabei in mehreren *PartitionPairs* vorkommen. Beispielweise kommt das Konzept 12755 aus Abbildung 3.5-1 in 5 Token vor. D. h. für jedes Token wird ein *PartitionPair* gebildet, und jedes *PartitionPair* wird das Konzept 12755 enthalten.

Algorithm *TokenBlocking*(O_1, O_2,)

Input: Two ontologies O_1 and O_2,
Output: *PartitionPairSet - set of Partition Pairs*

```
1:  PartitionPairSet ← {}
2:  //Domain
3:  mapsD ←buildTokenMaps(O1 , tokenMapD, tokenConceptsMapD )
4:  tokenMapD ← mapsD.getTokenMap()
5:  tokenConceptsMapD← mapsD.getTokenConceptsMap()
6:  //Range
7:  mapsR ← buildTokenMaps(O2 , tokenMapR, tokenConceptsMapR )
8:  tokenMapR ← mapsR.getTokenMap()
9:  tokenConceptsMapR ← mapsR.getTokenConceptsMap()
10: for each token ∈ tokenMapD do
11:     if tokenMapD.contains(token) && tokenMapR.contains(token) then
12:         domainPartition ← {}
13:         rangePartition ← {}
14:         for each cD ∈ tokenConceptsMapD.getConcepts(token) do
15:             for each cR ∈ tokenConceptsMapR.getConcepts(token) do
16:                 domainPartition.add(cD)
17:                 rangePartition.add(cR)
18:             end for
19:         end for
20:         PartitionPair ← buildPartitionPair(domainPartition, rangePartition)
21:         PartitionPairSet.add(PartitionPair)
22:     end if
23: end for
return PartitionPairSet
```

Algorithmus 11 TokenBlocking: mehrere PartitionPairs

3.6 KMeans-Tokenbasiertes Blocking

Das KMeans-Tokenbasierte Blocking führt zuerst, ähnlich dem *TokenBlocking*, eine Tokenisierung der Konzept-Attribute durch. Anstatt Konzepte anhand überlappender Token einem *PartitionPair* zuzuordnen, soll hier ein clusterbasieres Verfahren durchgeführt werden, um mit Hilfe des KMeans-Algorithmus die Cluster und danach die *PartitionPairs* zu bilden. Das *KMeansTokenBlocking* ist im Algorithmus 12 dargestellt.

Algorithm *KMeansTokenBlocking*(O_1, O_2,, *IM*)

Input: Two ontologies O_1 and O_2,
Output: *PartitionPairSet - Set of PartitionPairs*

```
1:  ProcessedDConcepts ← {}
2:  ProcessedRConcepts ← {}
3:  PartitionPairSet ← {}
4:  //Domain
5:  mapsD ←buildTokenMaps(O1 , tokenMapD, tokenConceptsMapD )
6:  tokenMapD ← mapsD.getTokenMap()
7:  tokenConceptsMapD← mapsD.getTokenConceptsMap()
8:  //Range
9:  mapsR ← buildTokenMaps(O2 , tokenMapR, tokenConceptsMapR )
10: tokenMapR ← mapsR.getTokenMap()
11: tokenConceptsMapR ← mapsR.getTokenConceptsMap()
12: mixedMap ← doMix(tokenMapD, tokenMapR)
13: Cluster[] ←KMeans(mixedMap)
14: for each Cluster ∈ Cluster[] do
15:    Blocks[]←buildBlocks()
16:    corrCount ←findCorrCount( Blocks[])
17:   if corrCount >= IM.size() / Cluster[] size() do
18:      PartitionPair ← buildPartitionPair(domainPartition, rangePartition)
19:      PartitionPairSet.add(PartitionPair)
20:   end if
21: end for
return PartitionPairSet
```

Algorithmus 12 KMeansTokenBlocking

Wie beim tokenbasierten Ansatz werden für jedes Konzept aus O_1 und O_2 die Attribute extrahiert und tokenisiert (siehe Abbildung 3.5-1). Dabei entstehen zwei Maps pro Ontologie: ein Map mit den Token und deren Vorkommens-Anzahl (<*T, F*>) und eine Zuordnung der Token zu den Konzepten, in denen diese Token vorkommen (<*T*, <*C*>>). Im nächsten Schritt werden die <*T, F*>-Maps vereinigt (Algorithmus 12, Zeile 12). Das entstehende Map enthält die Token, die in beiden Ontologien auftreten sowie dessen Frequenzen (Beispielweise kommt "system" 4-mal in O_1 und 6-mal in O_2 vor, so dass das Ergebnis folgendermaßen aussieht: {"system", {4,6}}). Dieses Map wird als Eingabe für

das KMeans-Clustering verwendet. Mit Hilfe des KMeans-Algorithmus werden die Token zu 10 Clustern mit zufällig ausgewählten Zentroiden zugeordnet. Falls die Cluster keine Elemente enthalten, werden sie nicht weiter berücksichtigt. Dies kann auf Grund einer zufälligen Auswahl der Zentroiden passieren. Aus den <*T*, <*C*>>-Zuordnungen werden entsprechende Konzepte extrahiert und in *Domain*- und *Range-Partitionen* aufgeteilt (siehe Abbildung 3.6-1).

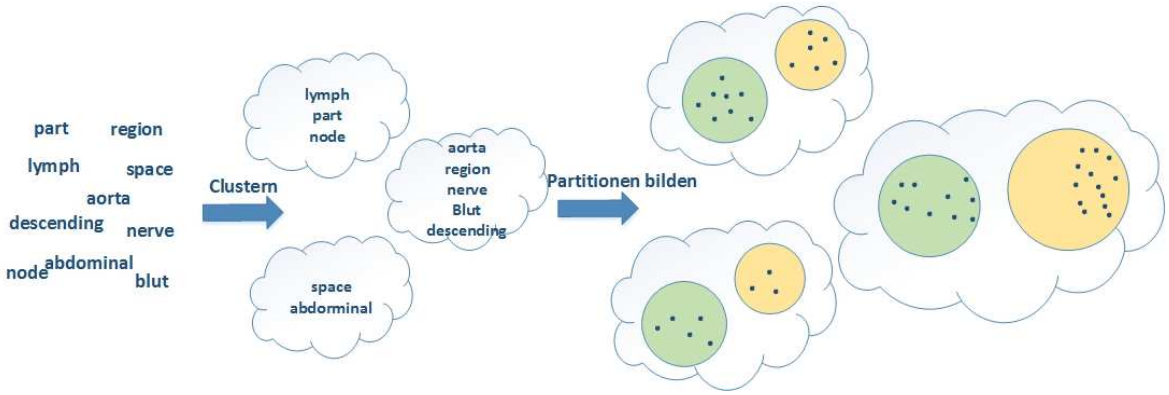


Abbildung 3.6-1 KMeans-Tokenbasierte Blocking: Cluster- und Partitionsbildung

Ein initiales Mapping (*IM*, siehe Kapitel 3.1) wird für die Wahl der Cluster für das spätere Matching eingesetzt. Falls der Korrespondenzanteil innerhalb eines Clusters über dem gegebenen Threshold liegt, wird das Cluster zur weiteren Bearbeitung verwendet, d. h. es wird ein *PartitionPair* erstellt. Angenommen, der Threshold wird auf 3 gesetzt, dann entspricht Cluster 2 in Abbildung 3.6-2 den Anforderungen, da es 4 Korrespondenzen aus *IM* zwischen *Domain*- und *Range-Partitionen* aufweist.

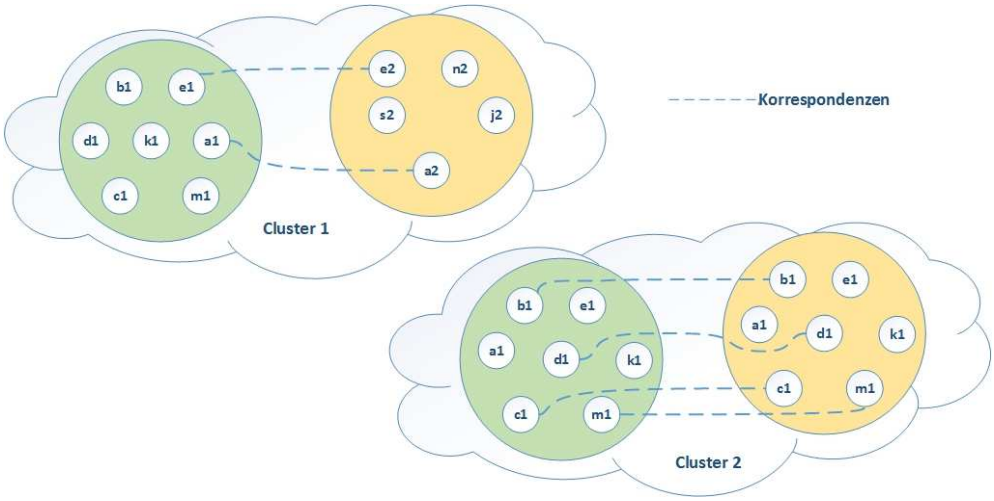


Abbildung 3.6-2 KMeans-Tokenbasiertes Blocking: Auswahl von Clustern

Jedes ausgewähltes Cluster bildet ein *PartitionPair*, das für ein weiteres Matching übergeben wird.

4 Evaluierung

4.1 Setup

Für die Evaluierung der Blocking-Verfahren werden die FMA, NCIT und SNOMED Ontologien (siehe Kapitel 2.3) abgeglichen und die Referenz-Mappings des OAEI *Large BioTracks* verwendet. Dieser stellt so genannte *Original UMLS*[9] und *Repaired UMLS* Datensätze (jeweils kleine und große Varianten) zur Verfügung. *Original UMLS*-Mappings werden durch die Extraktion von Mappings zwischen Ontologien anhand der Zuordnung von Ontologiekonzepten zum gleichen UMLS-Konzept gebildet. Dies wurde durch die *Large BioTrack*-Organisatoren realisiert. *Original UMLS*-Mappings ist nur ein Silberstandard, da die Referenz-Mappings nur aus *UMLS* extrahiert werden. Da Mappings logische Inkonsistenzen enthalten können, haben die *Large BioTrack*-Organisatoren die Mappings durch Anwendung der Systeme *LogMap*[14] und *Alcomo* [17] repariert. Somit entstanden *Repaired UMLS* Mappings. In den nächsten Kapiteln werden nur die *Original UMLS* Datensätze betrachtet. Die Auswertung für *Repaired UMLS*-Mappings befindet sich im Anhang C. Die Datensätze wurden zunächst in das GOMMA-Repository importiert, danach erfolgte Blocking, Matching und Auswertung der Ergebnisse anhand des in Kapitel 2.4 beschriebenen Workflows.

Im Rahmen dieser Masterarbeit werden drei Matchprobleme evaluiert: FMA-NCIT, FMA-SNOMED und SNOMED -NCIT, d. h. die jeweiligen Ontologien werden durch Anwendung der vorgestellten Blocking-Verfahren (siehe im Kapitel 3) partitioniert und anschließend gematcht. Tabelle 4.1-1 und die Tabelle 4.2-2 illustrieren, wie viele Konzepte die untersuchten Ontologien haben.

[9] Unified Medical Language System (UMLS) ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen und anderen Systemen: http://www.nlm.nih.gov/research/umls/.

Tabelle 4.1-1 Konzept-Anzahl: Original UMLS-Datensatz (small)

Ontologie-Fragmente	**Größe** (Konzept-Anzahl)
FMA (overlap NCIT)	3720
FMA (overlap SCT)	10181
NCI (overlap FMA)	6551
NCI (overlap SNOMED)	24039
SNOMED (overlap FMA)	13430
SNOMED (ovelap NCIT)	51176

Tabelle 4.1-2 Konzept-Anzahl: Original UMLS-Datensatz (whole)

Ontologie	**Größe** (Konzept-Anzahl)
FMA	79042
NCIT	66913
SNOMED	122516

Zur Bestimmung der Mappings zwischen den Ontologien wird immer der *Name-Synonym-Matcher* verwendet. Der *Name-Synonym-Matcher* berechnet die Ähnlichkeit zwischen Konzeptnamen und Synonymen. Um die Mappingqualität (Precision, Recall, F-Measure) bezüglich des Einflusses verschiedener Blocking-Verfahren zu messen, wird das Referenz-Mapping des *Large BioTrack* der OAEI 2012 verwendet. Die Anzahl der Korrespondenzen im Referenz-Mapping umfasst: FMA-NCI - 3024; FMA-SCT - 9008, SCT-NCI - 18840.

Es wurden verschiedene Einstellungen für das Cluster-Blocking untersucht. In Kapitel 4.2 wird nur die optimale Einstellung (Distanz = 2; ein Konzept kann nur zu einem Cluster zugeordnet werden; alle verbundenen Cluster werden als *PartitionPairs* betrachtet) mit den anderen Blocking-Verfahren verglichen. Die Ergebnisse des Cluster-Blockings für weitere getestete Einstellungen befinden sich im Anhang A. Da die Zentroide für das KMeans-Tokenbasierte Blocking zufällig bestimmt werden, werden 10 Durchläufe für alle Matchprobleme ausgeführt, um eine Aussage zur Stabilität dieses Blocking-Verfahrens zu ermöglichen (siehe Anhang B). In Kapitel 4.2 wird nur der Mittelwert dargestellt. Beim tokenbasieten Blocking wurden mehrere *PartitionPairs* gebildet.

Für die Evaluierung der Blocking-Verfahren wurde folgende Konfiguration des Rechners verwendet:

- Betriebssystem: Windows 7 Enterprise, Service Pack 1
- System-Typ: x64-basierter PC
- Prozessor: Intel Xeon CPU, 2,67 GHz, 4 Kerne, 8 logischer Prozessoren
- Installierter physikalischer Speicher: 4 GB
- Entwicklungsumgebung: Eclipse Juno, JavaSE-1.6, Java heap space 3GB

4.2 Analyse der Blocking-Verfahren

Die Analyse der Blocking-Verfahren erfolgt bezüglich der Ausführungszeit und der Qualität der zu ermittelnden Mappings. Dazu wird jeweils der gleiche Match-Workflow ausgeführt (siehe Kapitel 2.4), jedoch jeweils ein anderes Blocking-Verfahren verwendet.

4.2.1 Ausführungszeit

Um sich unnötige Vergleiche zwischen den Ontologien zu ersparen und dadurch die Ausführungszeit für das Matching zu reduzieren, werden verschiedene Blocking-Verfahren eingesetzt. Dieses Ziel wird aber nicht immer erreicht.

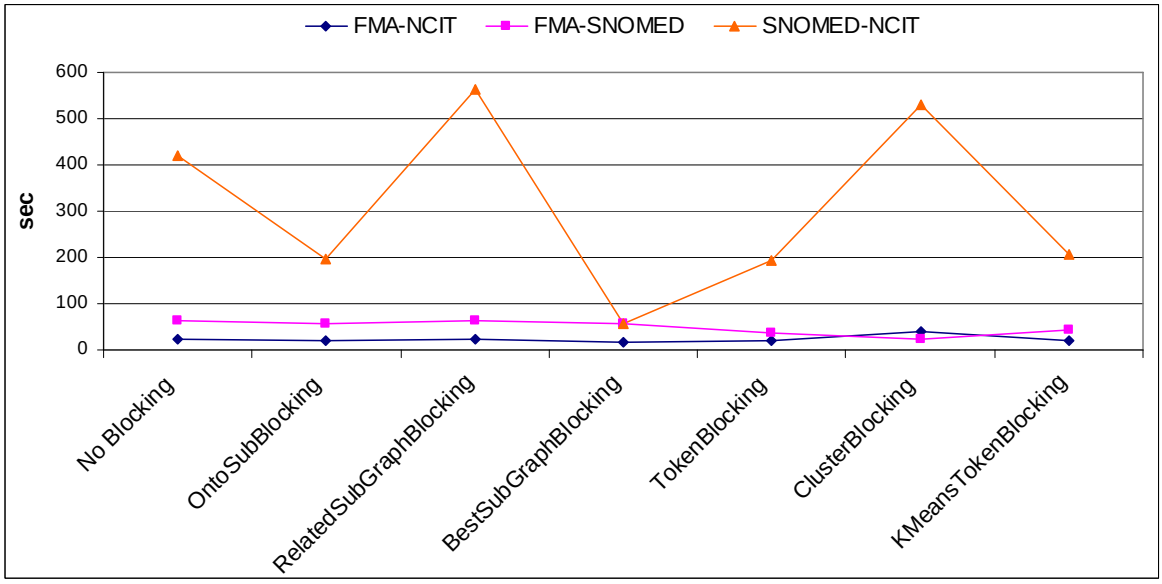


Abbildung 4.2-1 Ausführungszeit: Original UMLS small

Die Abbildung 4.2-1 und die Tabelle 4.2-1 zeigen die Zeitmessungen für die kleineren Ontologie-Varianten (small). Das Blocking anhand der am besten verknüpften Ontologie-Subgraphen (*BestSubGraphBlocking*) zeigt für alle drei Matchprobleme die geringste Laufzeit. In diesem Fall wurde das Matching für das Matchproblem SNOMED-NCIT sieben Mal schneller durchgeführt als ohne Blocking. Mit *OntoSubBlocking*, *TokenBlocking* und *KMeansTokenBlocking* werden ebenfalls gute Werte erreicht. Das clusterbasierte Blocking und das Blocking anhand der zusammenhängenden Ontologie-Subgraphen weisen im Gegensatz dazu eine deutlich schlechtere Ausführungszeit auf.

Tabelle 4.2-1 Ausführungszeit: Original UMLS small (in Sekunden)

Blocking Verfahren	FMA-NCIT	FMA-SNOMED	SNOMED -NCIT
No Blocking	25	65	422
OntoSubBlocking	18	57	197
RelatedSubGraphBlocking	23	63	563
BestSubGraphBlocking	18	56	58
TokenBlocking	21	37	195
ClusterBlocking	40	22	531
KMeansTokenBlocking	19	42	208

Ein wichtiges Ziel der Blocking-Verfahren ist es, die Ausführungszeit insbesondere für das Matching von sehr großen Ontologien zu reduzieren. Dabei spielen die Ontologiestruktur sowie die Ontologiegröße eine große Rolle. Für komplexere Matchprobleme wie FMA-SNOMED dauert die Bearbeitung ohne Blocking fast 40 Minuten.

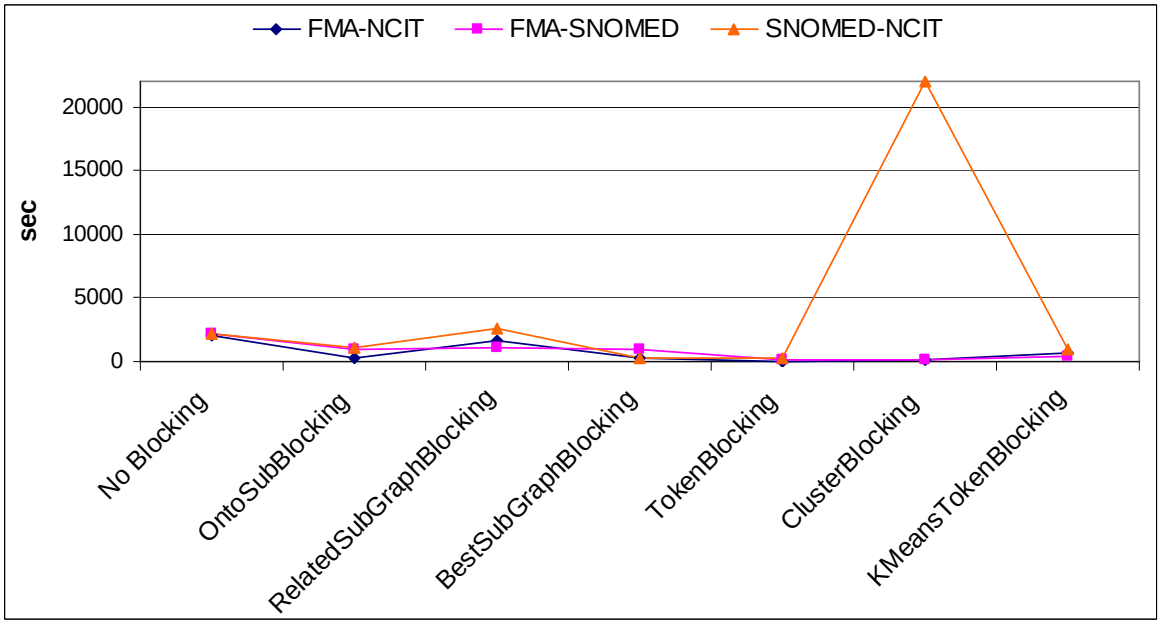


Abbildung 4.2-2 Ausführungszeit: Original UMLS whole

Abbildung 4.2-2 und Tabelle 4.2-2 zeigen die Ausführungszeiten für sehr große Ontologien (whole-Datensatz). Fünf Blocking-Ansätze ermöglichen eine Verbesserung der

Ausführungszeit gegenüber einer Ausführung ohne Blocking. Beispielweise erfolgt das Matching unter Verwendung von *TokenBlocking* 20-mal schneller (FMA-SNOMED) als ohne Blocking und liefert somit durchschnittlich das beste Ergebnis der Blocking-Verfahren für alle Matchprobleme bzgl. die beste Ausführungszeit. Die größte Zeitersparnis wird für FMA-SNOMED erreicht (von 37 Minuten (2225 Sekunden) ohne Blocking zu fast 1 Minute für *ClusterBlocking*). Das *TokenBlocking* ermöglicht, dass jedes Matchproblem in weniger als 5 Minuten gematcht werden kann. Für die Matchprobleme FMA-NCIT und FMA-SNOMED hat die Verwendung aller Blocker zeitliche Vorteile gebracht. Für SNOMED-NCIT waren die Ausführungszeiten von *RelatedSub-GraphBlocker* nicht besser und von *ClusterBlocker* deutlich schlechter als ohne Blocking.

Tabelle 4.2-2 Ausführungszeit: Original UMLS whole (Sekunden)

Blocking-Verfahren	FMA-NCIT	FMA-SNOMED	SNOMED -NCIT
No Blocking	2058	2225	2199
OntoSubBlocking	320	988	1094
RelatedSubGraphBlocking	1606	1107	2663
BestSubGraphBlocking	321	978	238
TokenBlocking	63	97	227
ClusterBlocking	145	74	21989
KMeansTokenBlocking	667	470	921

Um die Gründe der langsamen Verarbeitung unter Verwendung einiger Blocking-Verfahren zu untersuchen, soll die Ausführungszeitaufteilung analysiert werden (siehe Abbildung 4.2-3). Die gesamte Ausführungszeit besteht aus vier Teilen: Laden der Ontologien, Blocking-Prozess, Matching und Postprozessing. Das Ziel der Anwendung von Blocking-Verfahren ist es, die Anzahl der Vergleiche zu minimieren und dadurch die Zeit für das Matching zu verringern. An dieser Stelle werden einige Matchprobleme in Zusammenhang mit den Blocking-Verfahren, die zu einer Verschlechterung der Ausführungszeit führen, näher betrachtet.

Tabelle 4.2-3 Anzahl der Vergleiche

Matchproblem	Blocking-Verfahren	Anzahl der *PartitionPairs*	Anzahl der Vergleiche mit Blocking	Anzahl der Vergleiche ohne Blocking
FMA-NCIT small	OntoSubBlocking	1	11160000	24369720
	RelatedSubGraphBlocking	1	16048032	
	BestSubGraphBlocking	1	10740000	
	TokenBlocking	2003	43248	
	ClusterBlocking	781	7332262	
	KMeansTokenBlocking	3	9244602	
FMA-NCIT whole	OntoSubBlocking	1	922815350	5288937346
	RelatedSubGraphBlocking	1	4206189988	
	BestSubGraphBlocking	1	916977850	
	TokenBlocking	2657	537745	
	ClusterBlocking	845	73943152	
	KMeansTokenBlocking	2	1276813262	
FMA-SNOMED small	OntoSubBlocking	1	106992129	136730830
	RelatedSubGraphBlocking	1	106953210	
	BestSubGraphBlocking	1	105247635	
	TokenBlocking	2630	311561	
	ClusterBlocking	694	2112604	
	KMeansTokenBlocking	3	47547549	
FMA-SNOMED whole	OntoSubBlocking	1	4506105378	9683909672
	RelatedSubGraphBlocking	1	4523484784	
	BestSubGraphBlocking	1	4477600878	
	TokenBlocking	3511	1871648	
	ClusterBlocking	747	4145839	
	KMeansTokenBlocking	5	1008031011	
SNOMED-NCIT small	OntoSubBlocking	1	427822083	1230219864
	RelatedSubGraphBlocking	1	378751464	
	BestSubGraphBlocking	1	70458323	
	TokenBlocking	9103	330664	
	ClusterBlocking	4646	135894097	
	KMeansTokenBlocking	2	401323296	
SNOMED-NCIT whole	OntoSubBlocking	1	3814643217	8197913108
	RelatedSubGraphBlocking	1	3823551600	
	BestSubGraphBlocking	1	824122104	
	TokenBlocking	11402	676278	
	ClusterBlocking	4541	673887578	
	KMeansTokenBlocking	2	2490036360	

Tabelle 4.2-3 zeigt die Anzahl der durchgeführten Vergleiche. Bei der Verwendung der Blocking-Verfahren wird immer eine Reduzierung der Anzahl der Vergleiche erreicht. Nichtsdestotrotz hat das Matching mit vorherigem Blocking mehr als doppelt soviel Zeit gebraucht als ohne die Blocking-Anwendung. Für die Vereinigung der einzelnen Match-Teilergebnisse pro *PartitionPair (*Beispielweise SNOMED-NCIT whole: 4541*)* wurden

nur wenige Sekunden benötigt, was keine Auswirkung auf das Gesamtergebnis hat, d. h. der Grund für die lange Laufzeit liegt woanders. Aufgrund der teilweise hohen Anzahl von *PartitionPairs* (z.B. 4646, 4541) werden ebenso viele einzelne Match-Tasks innerhalb von GOMMA generiert (ein Aufruf des Name-Synonym-Matchers pro PartitionPair). Dies ist eine sehr generische workflow-artige Realisierung der Blocking-Verfahren in Kombination mit Matching, die jedoch negative Auswirkungen auf die Ausführungszeit haben kann, falls extrem viele einzelne Match-Tasks generiert werden.

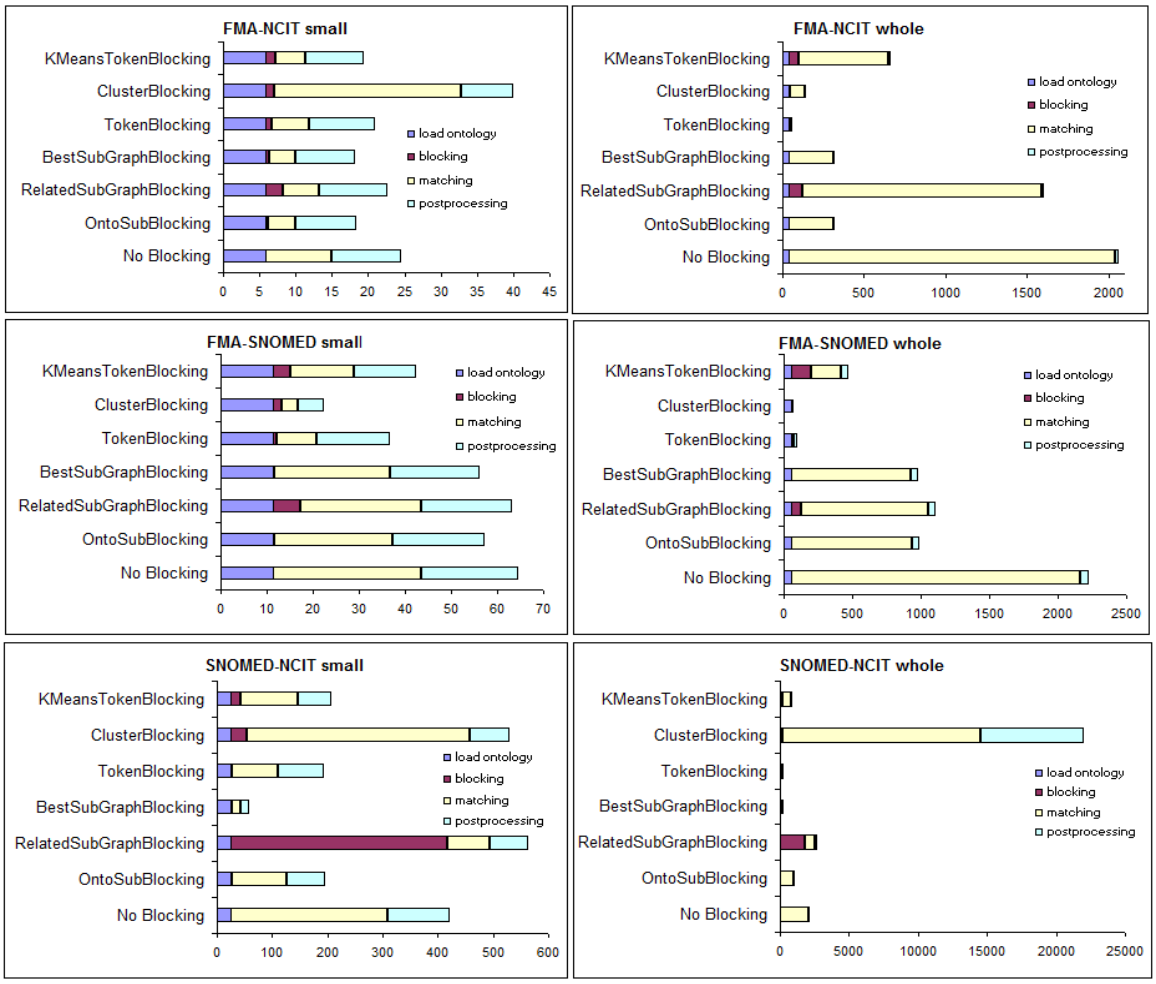


Abbildung 4.2-3 Ausführungszeitaufteilung

Jedoch beeinflusst nicht nur die Anzahl der Vergleiche die Gesamtausführungszeit. Auch die Ausführung des Blockings selbst kann teilweise lange dauern. Die Verwendung des *RelatedSubGraphBlocking* für das Matchproblem SNOMED-NCIT braucht genauso viel

Zeit, wie der komplette Ontologie-Vergleich ohne Blocking. Der Grund dafür kann in der komplexeren Struktur von SNOMED und der enormen Größe der Ontologie liegen.

4.2.2 Mappingqualität

Neben geringen Ausführungszeiten sollen auch Mappings von hoher Qualität bestimmt werden. Die Qualität der Mappings hängt unter anderem von dem betrachteten Matchproblem, der Ontologiegröße, den verwendeten Match-Verfahren sowie den hier angeführten Blocking-Verfahren ab. Zum Vergleich zum „herkömmlichen“ Matching wurde für jedes Matchproblem in jeder Größe ein Matchvorgang ohne Blocking durchgeführt. Dabei wurden die Precision-, Recall- und F-Measure-Werte bzgl. der Qualität des Mappings berechnet.

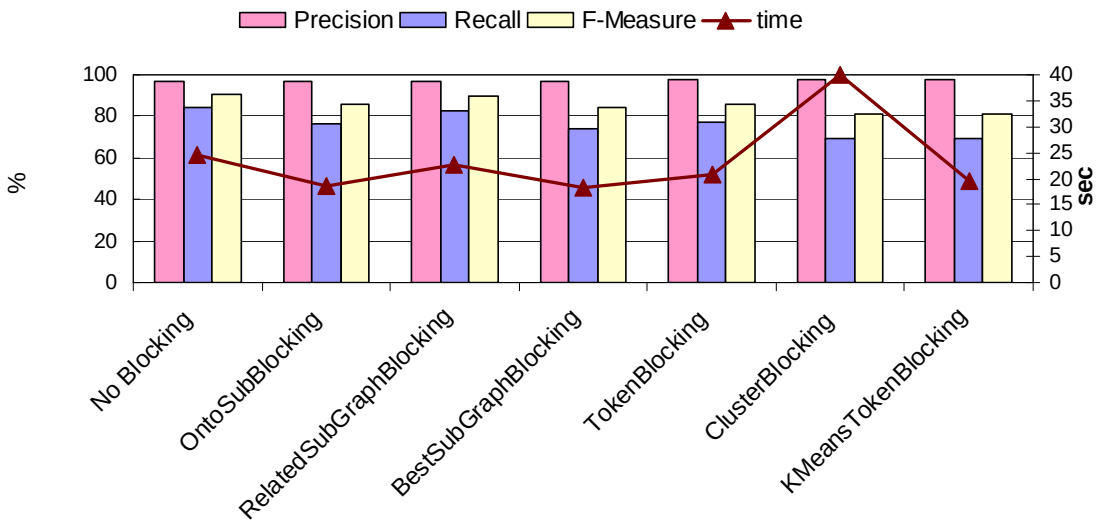


Abbildung 4.2-4 FMA-NCI small

Die Abbildung 4.2-4 zeigt die Ergebnisse für FMA-NCI small. Der Einsatz der sechs Blocking-Verfahren führt zu guten Precision-, Recall- und F-Measure-Werten (F-Measure von mehr als 80%), auch wenn die Werte im Vergleich zur Ausführung "ohne Blocking" nicht überschritten werden konnten. Die *RelatedSubGraphBlocking*-Anwendung erzeugt das beste Ergebnis (F-Measure = 89,4%), welches beinahe genauso gut wie das Ergebnis "ohne Blocking" (F-Measure = 90,4 %) ist.

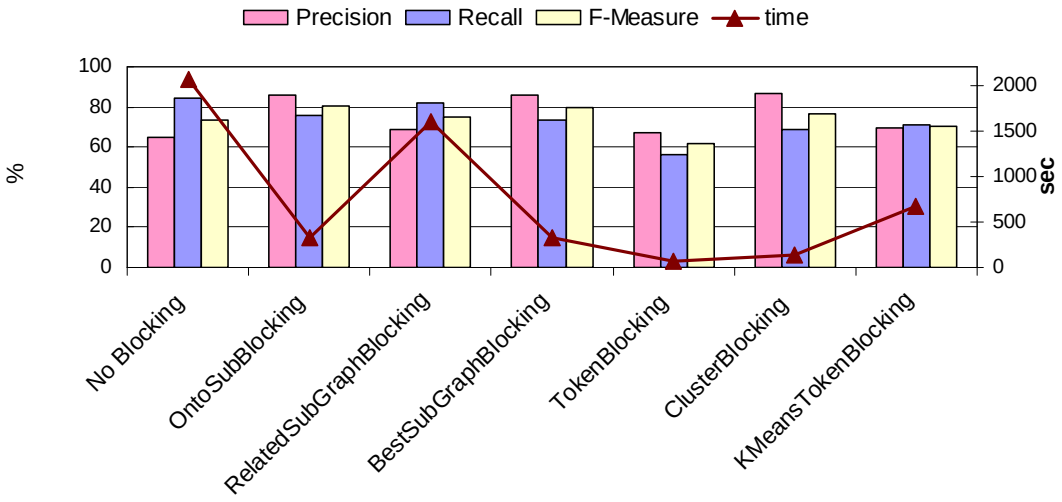


Abbildung 4.2-5 FMA-NCIT whole

Für FMA-NCIT whole (siehe Abbildung 4.2-5) unterscheiden sich die Ergebisse im Vergleich zum small-Datensatz. Unter Anwendung von vier Blocking-Verfahren (*OntoSubBlocker*, *ClusterBlocker*, *RelatedSubGraph-Blocker*, *BestSubgraphBlocker*) wurden bessere F-Measure-Werte als "ohne Blocking" erreicht. Dabei erfolgt das Matching unter Verwendung des *ClusterBlockers* in knapp 2,5 Minuten, was 14-mal schneller ist als "ohne Blocking".

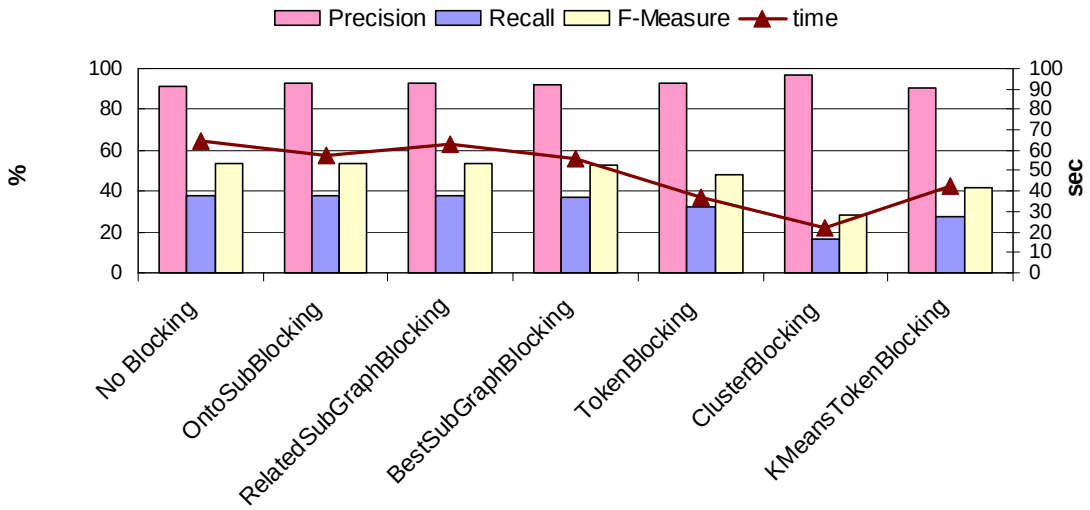


Abbildung 4.2-6 FMA-SNOMED small

Der Abbildung 4.2-6 ist zu entnehmen, dass trotz geringer Ausführungszeit und sehr hoher Precision das Gesamtergebnis mit *ClusterBlocking* für FMA-SNOMED (small) eine deutliche Verschlechterung bringt, da insbesondere das Recall stark sinkt. Die Verwendung von *OntoSubBlocking*, *RelatedSubGraphBlocking* und *BestSubgraphBlocking* liefert zwar ähnlich hohe F-Measure Werte wie "ohne Blocking", bringt aber fast keine zeitliche Verbesserung. Dies entspricht nicht dem Ziel des Blockings, und dessen Anwendung hat in diesem Fall keine – weder positiven noch negativen – Folgen. Interessant sind die Match-Ergebnisse für *TokenBlocking*. Die Ausführungszeit ist doppelt so schnell wie "ohne Blocking", wobei sich die Mappingqualität kaum verschlechtert.

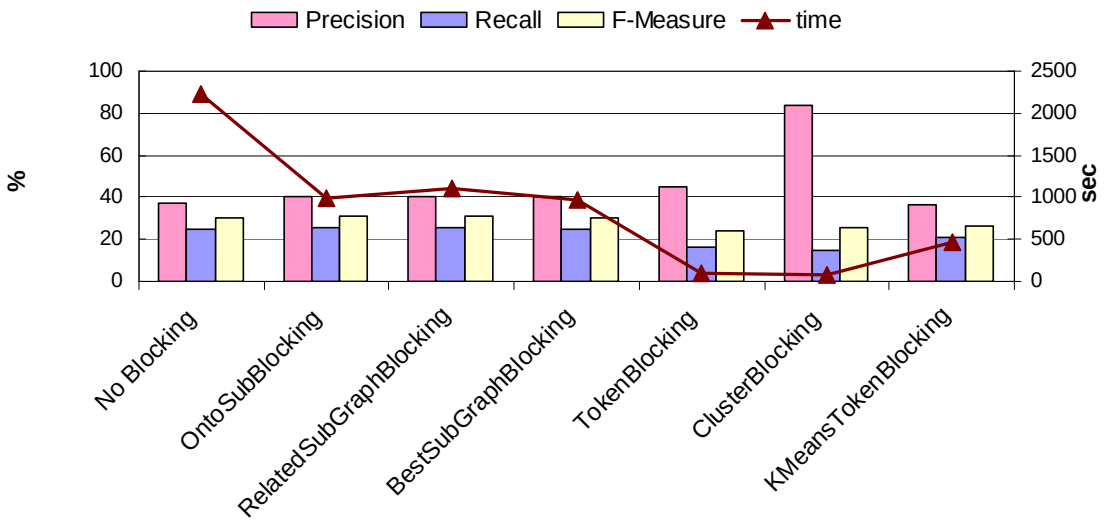


Abbildung 4.2-7 FMA-SNOMED whole

FMA-SNOMED (whole) (siehe Abbildung 4.2-7) stellt das größte Matchproblem dar, das in dieser Arbeit betrachtet wird. Der Einsatz von *OntoSubBlocking*, *RelatedSubGraph-Blocking* und *BestSubgraphBlocking* beschleunigt das Matching und sorgt dafür, dass die F-Measure Werte besser als "ohne Blocking" sind. Die anderen drei Blocker bringen zwar wesentlich mehr Zeitersparnis, verschlechtern jedoch gleichzeitig die Mappingqualität.

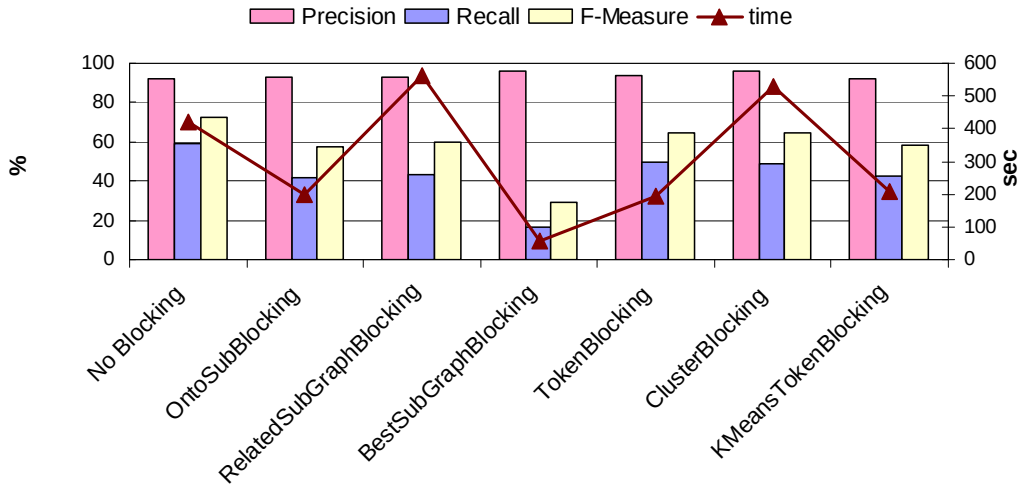


Abbildung 4.2-8 SNOMED-NCIT small

In Abbildung 4.2-8 und Abbildung 4.2-9 ist die Analyse der Mappingqualität für das SNOMED-NCIT Matchproblem dargestellt. Die schnellste Methode ist nicht immer die beste bezüglich der erreichten Mappingqualität. Die Verwendung des *BestSubGraphBlocking* führt zur schlechtesten Mappingqualität im Vergleich zu den anderen Blocking-Verfahren. Der Grund dafür ist, dass die Korrespondenzen über mehrere Subgraphen in den Ontologien verteilt sind und sich nicht in einem Subgraphen konzentrieren. Die beste Mappingqualität wird mit Hilfe des *TokenBlockings* und des *ClusterBlockings* erreicht.

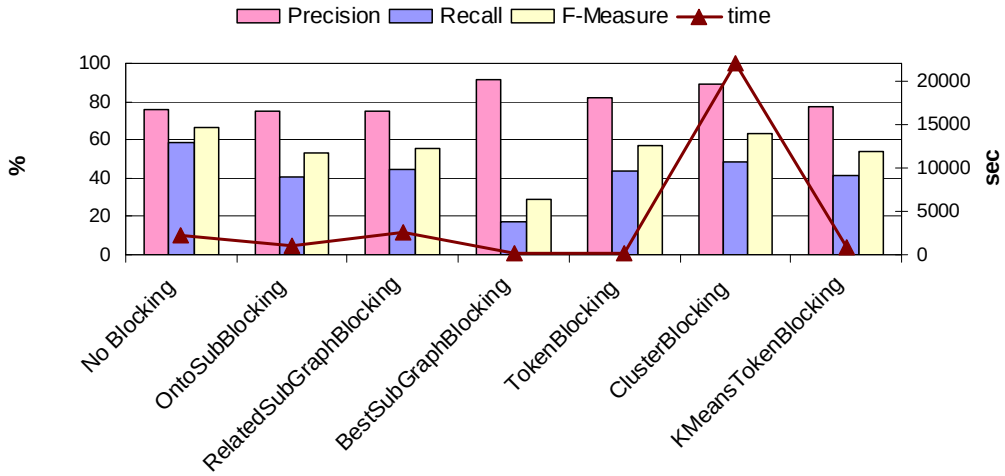


Abbildung 4.2-9 SNOMED-NCIT whole

Das Hauptziel der Verwendung von Blocking-Verfahren war die Reduktion des Suchraums für ein effizientes Matching von Ontologien. Die Ergebnisse zeigen, dass der Einsatz von Blocking-Verfahren für die kleinen Varianten der Matchprobleme weniger zeitliche Vorteile als für die großen Ontologien bringt. Dies ist naheliegend, da die kleinen Matchprobleme grundsätzlich weniger Zeit für den Vergleich benötigen und somit weniger Verbesserungspotential bieten. Die insgesamt größte Zeitersparnis wurde für FMA-NCIT erreicht (von 34 Minuten ohne Blocking auf fast 1 Minute unter Verwendung von *ClusterBlocking*). Das *TokenBlocking* führte insgesamt zu guten Ausführungszeiten, da jedes Matchproblem in weniger als 5 Minuten ausgeführt werden konnte. Da sich die Matchprobleme in ihrer Struktur und Größe unterscheiden, verhalten sich auch die Blocking-Algorithmen unterschiedlich. Die optimale Kombination zwischen Ausführungszeit und Mappingqualität bezüglich der untersuchten Verfahren zeigen folgende Blocker: FMA-NCIT – *OntoSubBlocking* und *BestSubGraphBlocking*; FMA-SNOMED – *OntoSubBlocking*, *BestSubGraphBlocking* und *RelatedSubGraphBlocking*; SNOMED-NCIT – *TokenBlocking*.

Die Evaluierung zeigte, dass zwar alle Blocking-Verfahren eine Reduktion der Anzahl von Vergleichen erreichen, jedoch nicht immer zu einem effizienteren Matching führen. So wurde durch das *ClusterBlocking* nur ein Neuntel der Vergleiche (SNOMED-NCIT whole) benötigt, jedoch brauchte das Matching mehr als doppelt so viel Zeit als ohne Blocking. Dies begründet sich vermutlich in der hohen Anzahl generierter *PartitionPairs* und somit einzeln zu verarbeitender Match-Tasks.

Darüber hinaus, kann eine Reduktion des Suchraums die Qualität der erzeugten Mappings beeinflussen. So bringt die Verwendung des *BestSubGraphBlockings* (SNOMED-NCIT whole) eine Verringerung der Anzahl notwendiger Vergleiche (Faktor 20 weniger), liefert jedoch gleichzeitig die schlechteste Mappingqualität im Vergleich zu den anderen Blocking-Verfahren. Dies liegt daran, dass die Korrespondenzen über mehrere Subgraphen in den Ontologien verteilt sind und sich nicht in einem Subgraph konzentrieren. Zudem zeigte sich, dass die Ausführung einiger Blocker selbst sehr lange dauert. Die Verwendung des *RelatedSubGraphBlocking* für das Matchproblem SNOMED-NCIT benötigte genauso

viel Zeit wie der komplette Ontologievergleich ohne Blocking. Dies begründet sich vermutlich in der komplexen Ontologiestruktur und der Ontologiegröße.

5 Zusammenfassung und Ausblick

Diese Arbeit beschäftigte sich mit der Etwicklung, Umsetzung und Evaluierung von Blocking-Verfahren zum effizienten Matching großer Ontologien in den Lebenswissenschaften. Im ersten Teil der Arbeit wurde dazu eine ausführliche Motivation gegeben. Die relevanten Grundlagen und verwandten Arbeiten wurden im zweiten Kapitel erläutert. Um die Analyse zu ermöglichen und die Robustheit, Effizienz und Effektivität der Anwendung von verschiedenen Blocking-Verfahren einzuschätzen, wurde der "Blocking von Ontologie-Subgraphen"-Ansatz [8] umgesetzt und weitere neuartige Verfahren vorgestellt. Folgende Blocking-Algorithmen wurden entwickelt und implementiert:

- Blocking anhand zusammenhängender Ontologie-Subgraphen (*RelatedSubGraph-Blocking*)

- Blocking anhand der am besten verknüpften Ontologie-Subgraphen (*BestSubGraphBlocking*)

- Tokenbasiertes Blocking (*TokenBlocking*)

- Clusterbasiertes Blocking (*ClusterBlocking*)

- KMeans-Tokenbasiertes Blocking (*KMeansTokenBlocking*)

Für die Evaluierung der Blocking-Verfahren wurden drei Matchprobleme für sehr große biomedizinische Ontologien untersucht: FMA-NCIT, FMA-SNOMED und SNOMED-NCIT.

Die Ergebnisse dieser Arbeit, haben gezeigt, dass die Verwendung von Blocking-Verfahren zur Reduktion des Suchraums und dadurch zu einer deutlichen Reduktion der Ausführungszeit führen kann. Es gibt keine optimale Lösung für alle Matchprobleme, aber

für jedes Matchproblem ist ein Blocking-Algorithmus vorhanden, der Mappings von hoher Qualität in geringerer Ausführungszeit im Vergleich zum Matching ohne Blocking erzeugt. Eine optimale Kombination zwischen Ausführungszeit und Mappingqualität bezüglich der untersuchten Verfahren zeigen folgende Blocking-Verfahren: FMA-NCIT – *OntoSubBlocking* und *BestSubGraphBlocking*; FMA- SNOMED – *OntoSubBlocking*, *BestSubGraphBlocking* und *RelatedSubGraphBlocking*; SNOMED-NCIT – *TokenBlocking*.

Die entwickelten Ansätze haben Stärken und Schwächen, die in zukünftigen Arbeiten näher untersucht werden sollen. Die Verfahren sollen neben den Lebenswissenschaften für weitere Domänen eingesetzt werden. Die vorgestellten Ansätze müssen entsprechend angepasst werden, damit sie stets stabile Resultate liefern. Dies gilt insbesondere für das *KMeansTokenBlocking* und dessen zufällige Auswahl von Zentroiden. Um eine weitere Zeitersparnis zu erreichen und die Mappingqualität des *TokenBlockers* zu verbessern, könnte die Anwendung einer anderen Art von Tokenisierung (Bildung von Wortpaaren) Vorteile bringen. Aufgrund der Generierung mehrerer *PartitionPairs* und zugehöriger Match-Tasks bietet es sich an, diese parallel auf mehreren CPUs und Rechnerknoten abzuarbeiten. Somit könnte eine weitere Reduktion der Laufzeit erreicht werden. Das GOMMA-Framework stellt bereits ein paralleles Matching zur Verfügung. Es ist sinnvoll, Blocking und Parallelisierung zu verknüpfen.

6 Literatur

1. Algergawy, A.; Massmann, S.; Rahm, E.: A Clustering-based Approach For Large-scale Ontology Matching Proc. ADBIS, 2011.

2. Bellahsene, Z.; Bonifati, A.; Rahm, E.: Schema Matching and Mapping. Springer-Verlag, 2011.

3. Bodenreider, O., Stevens, R.: Bio-ontologies: current trends and future directions. Briefngs in bioinformatics 7(3), 2006.

4. Coronado S. de, Haber M. W., Sioutos N., Tuttle M. S., and Wright L. W.: NCI Thesaurus: Using Science-based Terminology to Integrate Cancer Research Results. M. Fieschi, E. Coiera and Y-C.J. Li, editors. MEDINFO 2004. Proceedings of the 11th World Congress on Medical Informatics; 2004 Sep 7-11, San Francisco, CA, USA. Amsterdam: IOS Press, 2004.

5. Do, H.-H.; Rahm, E. Matching Large Schemas: Approaches and Evaluation, Information Systems, Volume 32, Issue 6, September, P. 857-885, 2007.

6. Euzenat, J.; Shvaiko P.: Ontology Matching, Springer, 2007 http://www.ncbi.nlm.nih.gov/pubmed/18467421

7. Groß, A.; Dos Reis, J.C.; Hartung, M.; Pruski, C.; Rahm, E.: Semi-Automatic Adaptation of Mappings between Life Science Ontologies Proc. 9th Intl. Conference on Data Integration in the Life Sciences (DILS), Montreal, July 2013.

8. Groß, A.; Hartung, M.; Kirsten, T.; Rahm, E.: GOMMA Results for OAEI 2012 Seventh International Workshop on Ontology Matching, 2012.

9. Groß, A.; Hartung, M.; Kirsten, T.; Rahm, E.: Mapping Composition for Matching Large Life Science Ontologies, 2nd International Conference on Biomedical Ontology, 2011.

10. Groß, A.; Hartung, M.; Kirsten, T.; Rahm, E.: On Matching Large Life Science Ontologies in Parallel, 7th International Conference on Data Integration in the Life Sciences, 2010.

11. Gruber: A translation approach to portable ontology specifications, Knowledge Acquisition, 1993.

12. Hamdi F, Safar B, Reynaud C, Zargayouna H.: Alignment-based Partitioning of Large-scale Ontologies. In: Advances in Knowledge Discovery And Management. Springer Studies in Computational Intelligence Series, 2009.

13. Hu W et al : Matching large ontologies: A divide-and-conquer-approach. Data Knowl. Eng. 67(1): 140-160, 2008.

14. Jimnez-Ruiz, E., Grau, B. C.: LogMap: Logic-Based and Scalable Ontology Matching. In: International Semantic Web Conference, 2011.

15. Kirsten, T.; Hartung, M.; Groß, A.; Rahm, E.: GOMMA: A Component-based Infrastructure for managing and analyzing Life Science Ontologies and their Evolution, Journal of Biomedical Semantics, Journal of Biomedical Semantics, 2011.

16. Maßmann, S.; Raunich, S.; Aümueller, D.; Arnold. P.; Rahm, E.: Evolution of the COMA match system In: ISWC Workshop on Ontology Matching, 2011.

17. Meilicke, C.: Alignment Incoherence in Ontology Matching. Ph.D. thesis, University of Mannheim, 2011.

18. Mungall, C., et al.: Uberon, an integrative multi-species anatomy ontology. Genome Biol 13(1), 2012.

19. Noy, N. F., Musen, M. A. Promptdiff: A fixed-point algorithm for comparing ontology versions. In Proceedings of the National Conference on Artificial Intelligence, P. 744-750, 2002.

20. Peukert E, Berthold H, Rahm E.: Rewrite Techniques for Performance Optimization of Schema Matching Processes. Proc. 13th Int. Conf. on Extending Database Technology, 2010.

21. Rahm, E., Bernstein, P. A. A survey of approaches to automatic schema matching. The VLDB Journal 10, 4, 2001.

22. Raunich, S.; Rahm, E.: Towards large-scale schema and ontology Proc. 7th OTM Workshop on Enterprise Integration, Interoperability and Networking, Springer LNCS, 2012.

Anhang A: Clusterbasiertes Blocking

		FMA-NCIT small	FMA-NCIT whole	FMA-SNOMED small	FMA-SNOMED whole	SNOMED - NCIT small	SNOMED-NCIT whole
	Precision	98,300	93,065	97,657	93,680	97,739	94,205
	Recall	61,177	61,243	14,809	14,809	45,653	45,642
	F-Measure	75,418	73,873	25,718	25,575	62,236	61,492
1	Gesamtzeit, Sek.	35,698	130,538	149,504	214,812	387,417	602,596
	Precision	97,441	86,341	97,045	83,941	96,258	88,841
	Recall	69,246	68,981	16,408	15,087	48,747	48,806
	F-Measure	80,959	76,691	28,070	25,576	64,719	63,001
2	Gesamtzeit, Sek.	41,814	198,679	221,820	296,360	830,697	22626,350
	Precision	97,276	82,353	95,851	76,387	95,815	95,815
	Recall	73,214	72,222	17,440	16,197	49,581	49,581
	F-Measure	83,547	76,956	29,511	26,727	65,347	65,347
3	Gesamtzeit, Sek.	39,864	233,794	261,335	358,061	966,346	41857,171
	Precision	98,105	92,779	97,732	93,557	97,497	94,012
	Recall	61,640	61,607	14,831	14,831	45,897	45,913
	F-Measure	75,711	74,046	25,754	25,604	62,413	61,695
4	Gesamtzeit, Sek.	40,020	135,264	154,474	220,062	407,580	655,783
	Precision	97,297	85,017	96,409	80,328	95,751	n/a
	Recall	72,619	72,619	17,285	16,319	49,878	n/a
	F-Measure	83,166	78,331	29,314	27,127	65,589	n/a
5	Gesamtzeit, Sek.	38,818	204,731	228,917	314,473	875,632	n/a
	Precision	97,314	80,493	94,776	68,823	95,283	n/a
	Recall	76,687	76,687	19,738	17,718	51,040	n/a
	F-Measure	85,778	78,544	32,672	28,180	66,473	n/a
6	Gesamtzeit, Sek.	39,427	302,538	342,869	506,582	1120,052	n/a
	Precision	96,952	88,427	97,191	92,927	97,678	94,340
	Recall	16,832	16,171	11,523	11,523	25,234	24,772
	F-Measure	28,684	27,341	20,603	20,504	40,106	39,240
7	Gesamtzeit, Sek.	12,671	63,977	80,846	143,339	216,151	295,483
	Precision	95,652	67,188	94,795	84,686	93,959	92,432
	Recall	2,183	1,422	7,682	6,139	22,866	4,538
	F-Measure	4,268	2,785	14,212	11,448	36,781	8,652
8	Gesamtzeit, Sek.	9399	59979	75396	134617	215120	264769
	Precision	95,588	75,439	94,661	79,521	93,501	94,614
	Recall	2,149	1,422	5,118	6,639	23,747	4,289
	F-Measure	4,204	2,791	9,710	12,254	37,875	8,206
9	Gesamtzeit, Sek.	9,207	59,927	74,922	136,094	221,339	270,717

1 - distance 1, ein Konzept kann nur einem Cluster zugeordnet werden, alle verbundenen Cluster werden gematcht
2 - distance 2, ein Konzept kann nur einem Cluster zugeordnet werden, alle verbundenen Cluster werden gematcht
3 - distance 3, ein Konzept kann nur einem Cluster zugeordnet werden, alle verbundenen Cluster werden gematcht
4 - distance 1, ein Konzept kann mehreren Clustern zugeordnet werden, alle verbundenen Cluster werden gematcht
5 - distance 2, ein Konzept kann mehreren Clustern zugeordnet werden, alle verbundenen Cluster werden gematcht
6 - distance 3, ein Konzept kann mehreren Clustern zugeordnet werden, alle verbundenen Cluster werden gematcht
7 - distance 1, ein Konzept kann mehreren Clustern zugeordnet werden, ein Cluster kann nur mit einem anderen Cluster verglichen werden
8 - distance 2, ein Konzept kann mehreren Clustern zugeordnet werden, ein Cluster kann nur mit einem anderen Cluster verglichen werden
9 - distance 3, ein Konzept kann mehreren Clustern zugeordnet werden, ein Cluster kann nur mit einem anderen Cluster verglichen werden

Anhang B: KMeans-Tokenbasiertes Blocking

		FMA-NCIT small	FMA-NCIT whole	FMA-SNOMED small	FMA-SNOMED whole	SNOMED-NCIT small	SNOMED-NCIT whole
	Precision	97,781	68,939	91,462	35,894	92,408	76,325
	Recall	74,306	73,909	32,582	22,147	42,834	42,489
1	F-Measure	84,442	71,337	48,048	27,393	58,536	54,589
	Precision	97,505	68,651	91,104	35,888	91,847	76,411
	Recall	74,967	74,735	32,971	22,402	42,696	43,105
2	F-Measure	84,764	71,564	48,419	27,585	58,294	55,117
	Precision	97,479	68,914	91,005	36,142	92,023	76,673
	Recall	75,430	74,041	31,672	22,380	43,227	43,355
3	F-Measure	85,048	71,385	46,990	27,643	58,823	55,389
	Precision	97,783	69,238	91,600	36,370	92,106	76,974
	Recall	78,770	75,397	34,503	22,336	43,726	43,880
4	F-Measure	87,253	72,186	50,125	27,675	59,300	55,896
	Precision	97,449	69,263	91,479	36,391	92,401	77,538
	Recall	74,537	72,057	32,538	21,670	43,116	42,930
5	F-Measure	84,467	70,632	48,002	27,164	58,796	55,263
	Precision	97,621	69,740	91,505	36,557	92,355	76,679
	Recall	69,213	71,032	32,405	20,560	42,192	42,235
6	F-Measure	80,998	70,380	47,860	26,318	57,923	54,468
	Precision	97,615	69,637	92,281	37,476	92,575	78,177
	Recall	73,082	71,065	32,249	21,625	42,882	42,479
7	F-Measure	83,585	70,344	47,795	27,425	58,614	55,047
	Precision	97,795	68,671	92,031	38,227	93,065	76,539
	Recall	71,858	71,759	32,948	22,258	42,452	42,097
8	F-Measure	82,844	70,181	48,524	28,134	58,307	54,318
	Precision	98,026	68,986	90,787	30,764	86,617	66,345
	Recall	32,837	42,956	23,190	13,865	11,200	17,861
9	F-Measure	49,195	52,945	36,944	19,115	19,835	28,145
	Precision	97,865	69,343	91,621	36,239	92,285	78,157
	Recall	37,897	73,677	31,805	20,859	42,224	42,277
10	F-Measure	54,636	71,445	47,219	26,478	57,939	54,872

Anhang C: Repaired UMLS Datensätze

	FMA-NCIT small	FMA-NCIT whole	FMA-SNOMED small	FMA-SNOMED whole	SNOMED-NCIT small	SNOMED-NCIT whole
OntoSubBlocking						
Precision	95,318	84,448	83,439	36,398	91,214	73,316
Recall	77,985	77,571	37,455	25,237	41,598	40,932
F-Measure	85,785	80,863	51,702	29,807	57,139	52,535
Vergleiche	11160000	922815350	106992129	4506105378	427822083	3814643217
Gesamtzeit, Sek.	20,05	327,591	56,494	983,844	196,003	1080,421
RelatedSubGraphBlocking						
Precision	94,641	66,768	83,452	36,477	91,345	73,697
Recall	84,092	83,678	37,492	25,274	43,962	44,634
F-Measure	89,055	74,273	51,740	29,859	59,357	55,596
Vergleiche	16048032	4206189988	106953210	4523484784	378751464	3823551600
Gesamtzeit, Sek.	22,797	1604,792	63,782	1096,207	543,284	2665,993
BestSubGraphBlocking						
Precision	95,405	84,615	83,166	35,826	94,916	90,266
Recall	75,949	75,535	36,790	24,572	17,223	17,213
F-Measure	84,573	79,818	51,013	29,150	29,156	28,912
Vergleiche	10740000	916977850	105247635	4477600878	70458323	824122104
Gesamtzeit, Sek.	18,864	315,863	56,285	988,857	58,714	240,653
TokenBlocking						
Precision	94,593	65,360	83,499	41,277	92,253	80,876
Recall	77,881	56,970	32,068	16,336	49,929	44,071
F-Measure	85,428	60,878	46,339	23,408	64,792	57,053
Vergleiche	43248	537745	311561	1871648	330664	676278
Gesamtzeit, Sek.	28,646	73,452	37,227	90,472	184,929	223,196
ClusterBlocking						
Precision	95,533	84,561	89,100	77,641	94,833	87,469
Recall	70,842	70,497	16,730	15,497	49,394	49,421
F-Measure	81,355	76,891	28,171	25,838	64,956	63,158
Vergleiche	7332262	73943152	2112604	4145839	135894097	673887578
Gesamtzeit, Sek.	39,52	154,492	22,298	68,631	513,763	21251,166
KMeansTokenBlocking						
Precision	96,305	68,336	81,919	28,608	87,167	67,775
Recall	40,476	46,618	28,209	16,422	21,618	22,148
F-Measure	56,997	55,426	41,966	20,866	34,644	33,385
Vergleiche	4151840	420177662	30986947	1340019920	205683960	1453714033
Gesamtzeit, Sek.	19,272	294,874	37,164	550,489	108,651	550,353
ohneBlocking						
Precision	94,406	62,790	82,458	33,394	90,869	74,681
Recall	85,611	85,128	38,133	25,028	59,870	59,193
F-Measure	89,794	72,272	52,150	28,612	72,182	66,041
Vergleiche	24369720	5288937346	136730830	9683909672	1230219864	8197913108
Gesamtzeit, Sek.	24,434	2033,541	67,351	2193,245	411,739	2102,857

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig 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.

Leipzig, 27.02.2014 Anastasiya Chyhir