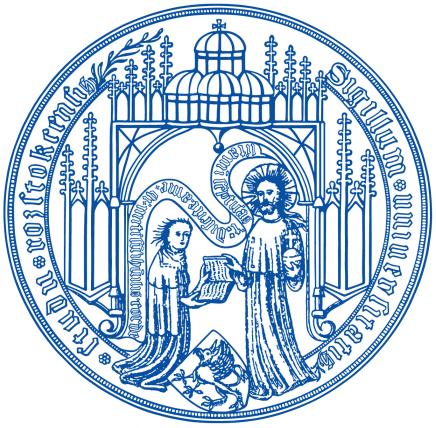
Parallelisierte Verarbeitung von Hypergraphstrukturen

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von: Matthias Kuhr
Matrikelnummer: 212207426
geboren am: 07.04.1993 in Rostock
Erstgutachter: Prof. Dr. Andreas Heuer
Zweitgutachter: Dr. Holger Meyer
Betreuer: Dr. Holger Meyer
Abgabedatum: 20. November 2017

SELBSTSTÄNDIGKEITSERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 20. November 2017

ZUSAMMENFASSUNG

Hypergraphen stellen ein wichtiges Werkzeug in vielen Anwendungsgebieten dar. Dazu zählen Schaltungsdesign für Chips, Parallelisierung von Berechnungen und auch Bereiche des maschinellen Lernens. Ein zentrales Problem stellt dabei die Suche nach einer guten Zerlegung (Partitionierung) eines Hypergraphen dar. Da die Anwendungsfälle oft große Hypergraphen mit sich bringen, wurden für entsprechende Verfahren auch parallelisierte Varianten entwickelt [Dev+06; TK08]. Partitionierungsstrategien, die eine verteilte Speicherung und parallele Auswertung des Hypergraphen selbst zum Ziel haben, sind hingegen kaum untersucht worden [HZY15].
Diese Arbeit analysiert die Übertragbarkeit bestehender Techniken auf dieses Problem im Kontext relationaler Datenbanken und anhand eines konkreten Anwendungsfalles. Dazu werden verschiedene Strategien vorgestellt und diskutiert. Diese umfassen Art (Vertex-, Edge-, Hybrid-Cut) der Zerlegung sowie Aspekte der Balancierung, Replikation und Minimierung der Schnittstellen einer Aufteilung. Der herkömmliche Ansatz der Knotenpartitionierung stellt sich dabei sowohl aus theoretischer als auch aus praktischer Sicht als weniger geeignet heraus. Eine Aufteilung der Hyperkanten führt im Falle des vorliegenden WossiDiA-Hypergraphen zu deutlich besseren Ergebnissen. Konzeptionelle Betrachtungen stellen gute theoretische Eigenschaften eines Hybrid-Cuts heraus, ein praktischer Vergleich mit den Resultaten des Vertex- und Edge-Cuts bleibt jedoch als eine interessante offene Fragestellung.

ABSTRACT

Hypergraphs are an essential tool in many areas, like VLSI design, parallel scientific computing or machine learning. An important and reoccurring task is to find a good partitioning for a given hypergraph. In order to deal with large hypergraphs, parallel algorithms for partitioning have been developed [Dev+06; TK08]. However, strategies, that aim to partition a hypergraph for its distribution itself, have not yet been extensively studied [HZY15].

This work aims to analyze, how existing solutions can be applied to this problem in the specific case of distributing the WossiDiA-hypergraph [MSS14] in a relational database. Therefore, multiple strategies are introduced and discussed. Those concern the type (vertex-, edge-, hybrid-cut) of the partitioning aswell as balance, replication and minimizing the resulting cut. Theoretical and practical results show, that a classic hypergraph-partitioning approach is less well suited for the problem than other methods. In a practical comparison a node partitioning performs significantly worse than an edge partitioning. While a hybrid-cut shows good theoretical properties, comparing it to the results of vertex- and edge-cuts remains an interesting open challenge.

INHALTSVERZEICHNIS

1 EINLEITUNG

Mit stetig anwachsenden Datenströmen aus verschiedensten Quellen werden Informationssysteme zunehmend mit großen Datenmengen konfrontiert. Über soziale Netzwerke, das Internet of Things oder autonome Fahrzeuge werden immer mehr Daten generiert. Damit wächst die Herausforderung, diese strukturiert abzuspeichern und für Analysen zugänglich zu machen. Ein mögliches Werkzeug dazu stellen Graphen und Hypergraphen dar, die sich insbesondere zum Speichern von Strukturen und Beziehungen zwischen Daten eignen. Vor allem Graphen sind über viele Anwendungsgebiete hinweg sehr verbreitet und auch in der Theorie umfassend erforscht. Daher gibt es für Graphen ein breites Spektrum an Werkzeugen und Plattformen, die eine effiziente Speicherung und Auswertung ermöglichen.

Hypergraphen hingegen sind eine Verallgemeinerung normaler Graphen und finden sich ebenfalls in einigen Anwendungsgebieten wieder. Dazu gehören klassische Anwendungen wie VLSI-Design [KK99], Datenbankmodellierung und Lastbalancierung [Dev+06], aber auch in den Bereichen Machine Learning [Zha+17b] oder digitale Archive [ABT12] kommen Hypergraphen zum Einsatz, so auch bei dem an der Rostocker Universität entwickelten WossiDiA System [MSS14]. Oftmals tauchen Hypergraphen als Datenstruktur für Berechnungen auf, in letzterem Anwendungsfall jedoch auch für die persistente Datenhaltung. Hypergraphen genießen nicht die gleiche Verbreitung wie Graphen und sind daher vor allem innerhalb ihrer Anwendungsgebiete gut verstanden.

Aufgrund der immensen Größe an zu verarbeitenden Datenmengen wird bei entsprechenden Anwendungsfällen statt auf einem einzelnen Rechner auf einen Verbund aus mehreren, parallel arbeitenden Rechnerknoten zurückgegriffen. Parallelität kann einen entscheidenden Vorteil bieten, wenn mehrere Rechnerknoten gleichzeitig arbeiten können. Dies gilt vor allem für die Verarbeitung großer Datenmengen, bei denen die Ausführungszeit einen wichtigen Faktor darstellt. Ein Beispiel dafür sind Selektionen auf Datenbanktabellen: Der Nutzer erwartet in der Regel Antwortzeiten im Millisekundenbereich, auch wenn die Relationen sehr groß sind. Um eine gute Skalierbarkeit mit wachsenden Datenmengen gewährleisten zu können, kann hier auf ein verteiltes Datenbanksystem (VDBMS) zurückgegriffen werden. Ein VDBMS verteilt sowohl die Daten als auch die Anfragen auf mehrere Rechnerknoten und nutzt die gewonnene Parallelität, um Anfragen schneller zu bearbeiten.

Diese Arbeit untersucht verschiedene Ansätze, um Hypergraphen verteilt in

einem VDBMS zu speichern und so komplexe Anfragen effizient verarbeiten zu können. Dazu werden verschiedene bestehende Techniken in den Bereichen Partitionierung und Parallelisierung für Graphen und Hypergraphen aber auch Datenbanksystemen beleuchtet. Aufgrund einer Vielzahl von Anwendungsfällen und deren spezifischen Anforderungen existieren bereits verschiedenste Strategien und Verfahren für typische Problemstellungen. Der konkrete Anwendungsfall einer Zerlegung von Hypergraphen mit dem Ziel einer parallelisierten Auswertung desselben gehört jedoch nicht dazu, sodass es sich hier um ein relativ unerforschtes Problem handelt.

Zentral ist dabei die Frage nach einer "guten" Partitionierung eines Hypergraphen, aus der eine Aufteilung der Daten in einem VDBMS abgeleitet werden kann und damit eine parallele Auswertung auf mehreren Datenknoten möglich wird. Was eine "gute" Verteilung ausmacht und wie eine solche erreicht werden kann, ist Gegenstand dieser Arbeit, die diese Frage speziell für das als Hypergraph in WossiDiA gespeicherte, digitalisierte Wossidlo-Archiv stellt. Dazu werden anhand der spezifischen Anforderungen und Eigenheiten der vorliegenden Datenstruktur verschiedene bestehende Techniken beleuchtet und hinsichtlich ihrer Übertragbarkeit auf das WossiDiA System untersucht. Damit soll eine konzeptionelle Grundlage geschaffen werden, mithilfe der sowohl eine erste Parallelisierung ermöglicht, jedoch auch zukünftige, sich ändernde, spezifische Anforderungen beantwortet werden können.

1.1 Aufgabenstellung

Das WossiDiA System bietet als digitales Archiv die Möglichkeit, komplexe Zusammenhänge auf verschiedenen Ebenen des Archivs zu beleuchten und zu untersuchen. Die große Menge der zu verarbeitenden Daten stellt dabei eine Herausforderung dar, da die nötigen Algorithmen oft mit erheblichem Rechenaufwand verbunden sind.

Um typische Anfragen effizient zu unterstützen, soll die Parallelisierung von Graphprimitiven (elementaren Graphalgebraoperationen) mittels eines verteilten Datenbanksystems untersucht werden. Darauf aufbauende, typische Anfrageoperationen sind k-Nachbarschafts-, k-Shortest-Path- und Synopsis-Operatoren im WossiDiA-System. Dazu müssen mögliche Verteilungsentwürfe und deren Ziele konzeptioniert sowie Techniken zur Umsetzung diskutiert werden. Ausgehend von der Untersuchung existierender Ansätze und Techniken sind Vorschläge für das WossiDiA-System zu entwerfen, diskutieren und prototypisch

umzusetzen.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit ist wie folgt aufgebaut. Kapitel 2 beleuchtet und motiviert den konkreten Anwendungsfall mit seinen spezifischen Anforderungen. Kapitel 3 stellt verteilte Datenbanksysteme mit Techniken und Hintergründen zur Verteilung und Parallelisierung vor. In Kapitel 4 werden Graphen und Hypergraphen sowie bestehende Techniken und Eigenschaften im Bereich der Graphpartitionierung vorgestellt. Kapitel 5 verbindet die in den vorangegangenen Abschnitten erarbeiteten Erkenntnisse aus Datenbank- und Graphtechniken mit den Anforderungen des Anwendungsfalls und diskutiert verschiedene Umsetzungsmöglichkeiten. Abschließend beschreibt Kapitel 6 Ergebnisse einer praktischen Umsetzung und zieht ein Fazit.

2 MOTIVATION

Die parallelisierte Verarbeitung von Hypergraphen ist eine wichtige Anwendung in vielen Bereichen. Hung et. al. konstatieren jedoch, dass "Hypergraphen zwar umfassend zur Optimierung verteilter Systeme eingesetzt werden, das Problem skalierbarer Berechnungen auf als Hypergraphen verteilt gespeicherten Daten selbst jedoch nicht erforscht wurde"[1] [HZY15]. Wie ein Ansatz auf Basis einer verteilten Datenbank aussehen kann, ist damit eine interessante offene Fragestellung. Dieser soll in der vorliegenden Arbeit speziell für das WossiDiA-System nachgegangen werden.

An der Universität Rostock wurde im Rahmen eines Forschungsprojektes das Wossidlo-Archiv digitalisiert und der Öffentlichkeit zugänglich gemacht. Dabei dient eine Hypergraphenstruktur, die Zusammenhänge zwischen den Entitäten abbildet, als Datenmodell. Im Folgenden soll zunächst der konkrete Anwendungsfall geschildert und auf die spezifischen Anforderungen eingegangen werden.

2.1 Das Wossidlo-Archiv

Richard Wossidlo war ein deutscher Volkskundler, der von 1859 bis 1939 in Mecklenburg-Vorpommern gelebt und gewirkt hat. Sein Lebenswerk ist das Wossidlo-Archiv, eine strukturierte Sammlung einzelner Fakten aus dem Altagsleben der Mecklenburger Bevölkerung. Dabei handelt es sich um Begriffe aus dem Arbeits- und Sozialleben, Bräuche, Lieder, Geschichten, Personen, Orte und vieles mehr. Diese wurden auf über zwei Millionen einzelner, kleiner Zettel erfasst und in einer hierarchischen Archivstruktur abgelegt. Mehrere Zettel wurden zu sogenannten Konvoluten anhand von Themen gruppiert und mit Meta-Informationen versehen. So entstanden ca. 28.000 solcher Konvolute, die in der sogenannten "Zettelwand" strukturiert abgelegt wurden. Abb. 1 zeigt den Feldforscher vor seiner Sammlung.

Wossidlo hat außerdem nicht nur die Fakten selbst erfasst, sondern auch von wann, wo und wem sie stammen. Dabei hatte Wossidlo mehrere Beiträger mit denen er korrespondierte und so die von ihnen übermittelten Informationen in das Archiv integrierte. Aus dieser Sammlung ist das Mecklenburger Wörterbuch entstanden, das die im Archiv enthaltenen Informationen nach Begriffen alphabetisch strukturiert zusammenfasst.

[1] Zitat frei aus dem Englischen übersetzt.

Abbildung 1: Richard Wossidlo vor seiner Zettelwand. Quelle: [MSS14]

Ein Unterschied des Wossidlo-Archivs zu anderen Archiven im Bereich der Volkskunde besteht darin, dass hier die Rohdaten der Feldforschung gesammelt und erhalten wurden, statt "nur" eine Gruppierung nach einem bestimmten Schema wie z.B. in einem Wörterbuch. Dadurch kann nicht nur nachvollzogen werden, wie Einträge aus dem Mecklenburger Wörterbuch zustande kommen, sondern die einzelnen Informationen und Zusammenhänge aus dem Archiv können nach verschiedenen Schemata neu gruppiert und ausgewertet werden. Statt einer Ordnung nach ontologischem Schema kann beispielsweise der geographische oder zeitliche Zusammenhang betrachtet werden. Für Volkskundler bietet sich damit die Möglichkeit, neue Strukturen und Beziehungen aufzudecken und so Fragestellungen wie "Wo entstehen Bräuche? Wie verändern sich Erzählungen? Welche Unterschiede in der Sprache gibt es zwischen sozialen Schichten?" auf den Grund zu gehen.

Bisher musste sich die Forschung bei solchen Aufgaben jedoch oft auf kleinere Gebiete beschränken, wie z.B. Genres, da die Quellen, auch wenn sie in Archiven strukturiert vorliegen, langwierige Recherchen mit sich bringen [ABT12]. Dies gilt insbesondere für das Wossidlo-Archiv, da hier die Informationen oft über viele Zettel an unterschiedlichen Stellen im Archiv verteilt sind. Will man etwa nachvollziehen, wie ein Eintrag im Mecklenburger Wörterbuch zustande gekommen ist, stellt das bei über zwei Millionen Zetteln eine wesentliche Herausforderung dar. Um das fragmentierte Wissen im Archiv wieder zusammen-

fügen und damit besser nutzen zu können, wurde das WossiDiA-Projekt mit Unterstützung der DFG ins Leben gerufen.

2.2 Das WossiDiA-System

Das WossiDiA-System [MSS14] hat zum einen die Aufgabe, die Forschungssammlung des Wossidlo-Archives öffentlich über das World Wide Web zugänglich zu machen. Außerdem wurde der digitalisierte Archivbestand zur Langzeitsicherung auf 35-mm Film ausbelichtet. Zum anderen dient es als Erfassungstool für die Bestände des Archivs und bietet eine Oberfläche, um Inhalte sowie entsprechende Metadaten einzupflegen. Dies beinhaltet nicht nur die Einträge auf den Zetteln, sondern auch die Details der Archivtopologie (Zettel-Konvolut-Box). Dafür wird dem Nutzer ein Typsystem zur Verfügung gestellt, um Einträge beispielsweise als Ort oder Person zu kennzeichnen. Im Hintergrund arbeitet eine Datenbank, in der alle Daten gespeichert werden.

Als Datenmodell wird ein typisierter Hypergraph verwendet, Abb. 2 zeigt einen Beispielgraphen. Einträge wie Orte, Zeiten oder Personen nehmen dabei die Rolle der Knoten ein. Jeder Knoten hat einen Knotentyp und ist je nach Typ mit verschiedenen Informationen angereichert. Ein Ort hat beispielsweise einen Namen sowie geographische Koordinaten. Im Beispiel finden sich die Orte “Güstrow” und “Rostock”, die Person “Ahrens”, sowie verschiedene Zeitspannen und Berufe.

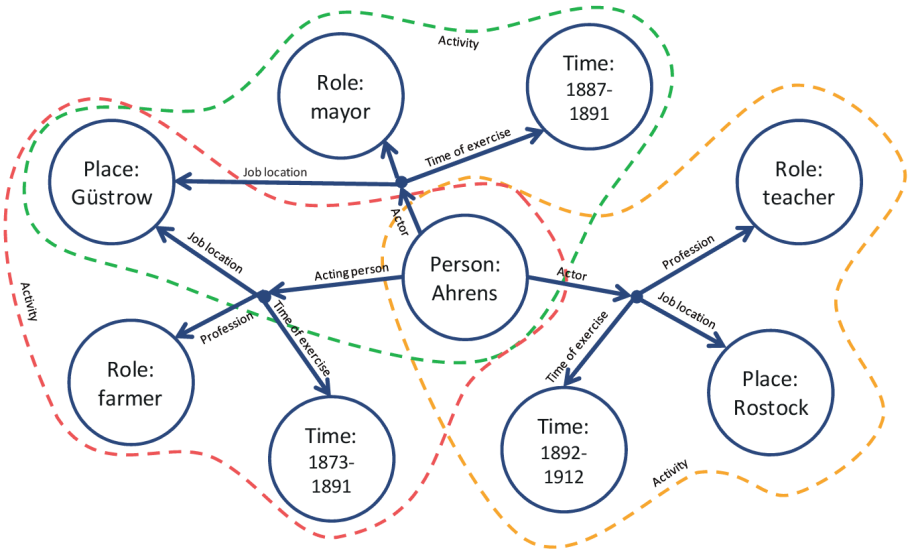


Abbildung 2: Quelle: [MSS14]

Die Zusammenhänge zwischen den Knoten werden durch Kanten modelliert. Im Beispiel sehen wir drei Kanten, die jeweils eine Aktivität repräsentieren. Die gelb gestrichelte Kante steht für das Faktum, dass die Person "Ahrens" von 1892 bis 1912 Lehrer in Rostock gewesen ist. Zusätzlich zur Zugehörigkeit zu einer Kante wird auch eine Rolle definiert, in der ein Knoten an einer Kante teilnimmt. Dadurch kann unterschieden werden, welche Funktion ein Knoten in einem Faktum einnimmt. Möchte man beispielsweise darstellen, dass Person A eine Geschichte an Person B erzählt hat, muss angegeben werden, wer Erzähler und wer Zuhörer ist. So gibt der Pfeil auf den Knoten "Rostock" in der gelben Kante an, dass dies der Arbeitsort ist. Das Datenmodell erlaubt einem Knoten damit auch, mehrfach in verschiedenen Rollen an einer Kante teilzunehmen.

Das vorliegende Datenmodell kann also durch seine Expressivität heterogene Daten sowie Beziehungen zwischen diesen abbilden. Daraus leiten sich für die Parallelisierung und auch für die Verteilungsstrategien besondere Herausforderungen ab. Zum einen kann mit der Typisierung eine weitere Quelle struktureller und semantischer Informationen neben der Kantenstruktur des Hypergraphen genutzt werden. Die Vielseitigkeit der Knoten und deren Beziehungen untereinander bedeutet aber auch eine erhöhte Komplexität bei der Auswertung von Strukturen. Betrachtet man beispielsweise nur Kanten, die die Archivtopologie modellieren, findet sich eine Baumstruktur. Ein anderes Muster zeigt sich, wenn man sich allein das Netzwerk von Beiträgern anschaut. Algorithmen, die auf dem gesamten Hypergraphen arbeiten, um z.B. eine Zerlegung dessen zu bestimmen, werten beide diese Strukturen undifferenziert voneinander aus. Dies gilt es bei einer Verteilungsstrategie und deren Umsetzung zu beachten.

2.3 Anwendungen und Ziele

Ein Anwendungsgebiet ist die Analyse von Erzählungen und deren Überlieferung. Welche Muster lassen sich darin erkennen, wie Geschichten, Märchen, Sagen und Legenden weitererzählt werden? Wie unterscheidet sich dies in verschiedenen sozialen Schichten oder Gruppen? Welche Rückschlüsse können damit auf Normen und Werte und deren Wandel gezogen werden? [ABT12] Solchen und ähnlichen Fragestellungen kann dank der strukturierten Speicherung als Hypergraph in WossiDiA durch entsprechende Analysen nachgegangen werden. Wie solche Anfragen jedoch genau aussehen und welche Zusammenhänge häufig ausgewertet werden sollen, ist für das Wossidlo-Archiv speziell aber auch andere Sammlungen selbst noch Gegenstand aktueller Forschungen. Daher geht diese Arbeit von grundlegenden Operationen aus, anstatt auf eine spezielle Ausrich-

tung von Analysen aufzubauen.

Solche Analysen werden zunächst in Form von Anfragen an die Datenbank vorgenommen. Praktische Vergleiche haben gezeigt, dass Datenbanken bei Aufgaben, die sich gut auf Datenbankoperationen abbilden lassen, anderen Systemen wie z.B. Hadoop durchaus überlegen sein können [Sto+10]. Komplexe Berechnungen, die Teile des Graphen auswerten und stark von einer optimierte Datenstruktur sowie speziellen Algorithmen profitieren, nutzen die Datenbank daher zumindest als Vorfilter.

Typische Anfragen durchsuchen den Hypergraphen beispielsweise nach einer vorgegebenen Kantenstruktur oder gehen der Frage, wie stark zwei Themenbereiche zusammenhängen, durch eine Auswertung der k kürzesten Pfade nach. Darüber hinaus sind Aggregationen über viele Knoten und Kanten wichtige Operationen. So ließe sich zum Beispiel feststellen, wie viele Personen in einem bestimmten Zeitraum mit einem oder mehreren Themen in Verbindung stehen. Solche Operationen sind jedoch oft mit nicht unerheblichem Rechenaufwand verbunden, insbesondere, da der zu verarbeitende Hypergraph groß ist. Um dieser Rechenlast zu begegnen und Analysen auch auf einem wachsenden Datenbestand möglich zu machen, soll die Anfrageverarbeitung parallelisiert werden.

Das Ziel besteht darin, eine paralleliserte Anfrageverarbeitung innerhalb der Datenbank zu nutzen und so Auswertungen zu beschleunigen. Diese ist für den Nutzer transparent, was den höheren Anwendungsschichten von WossiDiA ermöglicht, weiterhin ohne Anpassungen auf den Datenbestand zuzugreifen. Darüber hinaus besteht keine Bindung an spezielle Algorithmen oder Datenstrukturen, die Bandbreite der auch bisher möglichen Anfragen bleibt erhalten. Dies gilt insbesondere auch für Änderungsoperationen, die jederzeit möglich sein müssen. Um dieses Ziel zu erreichen, muss der Datenbestand und damit der Hypergraph verteilt werden.

Einen ersten Überblick über den zu verteilenden Datenbestand gibt Tabelle 1, aufgeschlüsselt nach Typen. Dabei sind Knoten und Kanten, die die Archivtopologie abbilden, nicht enthalten, was die Diskrepanz zu der deutlich größeren Anzahl von Zetteln erklärt. Wie in anderen Anwendungsfällen gilt auch hier, dass der Hypergraph mehr Kanten als Knoten aufweist.

Die Zugehörigkeit eines Knotens zu einer Kante wird in Form eines Links dargestellt, in dem Knoten- und Kanten-ID sowie Richtung und Rolle der Zugehörigkeit gespeichert werden. Abb. 3 stellt eine vereinfachte Sicht auf das Da-

Knoten		Kanten	
Anzahl	Knotentyp	Anzahl	Kantentyp
8.753	Person	14.225	Erzählt
4.901	Place	1.863	Herkunft
4.674	Event	5.309	Aktivität
41.399	Word	15.453	Synonym
197.741	Andere	302.982	Andere
257.468	insgesamt	339.832	insgesamt

Tabelle 1: Anzahl und Typ von Knoten und Hyperkanten in WossiDiA.

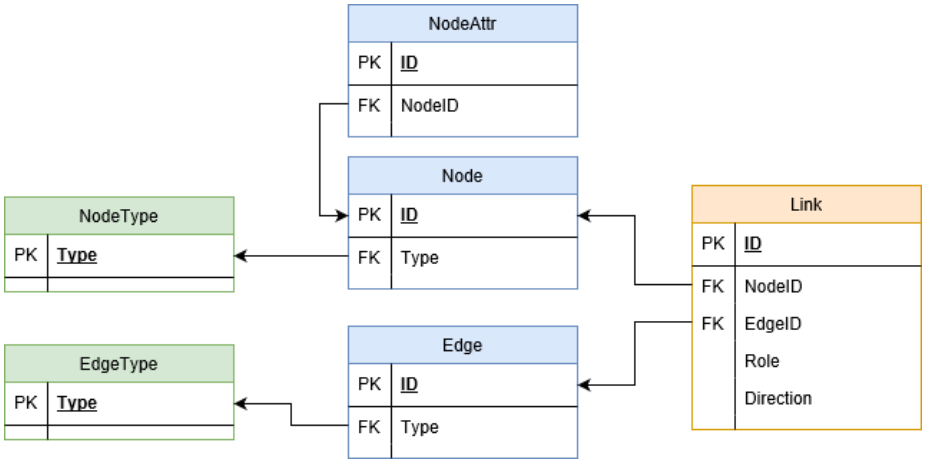


Abbildung 3: Eine Vereinfachte Sicht auf das Datenbankschema. Orange gekennzeichnete Tabellen bilden die Topologie, grün gefärbte die Typologie ab. Blau gekennzeichnete Tabellen halten Inhalte vor.

tenbankschema dar. Neben Knoten, Kanten und Links gibt es durch Attribute die Möglichkeit, diese näher zu beschreiben und so zusätzliche Informationen abzuspeichern. Solche Attribute werden in den Relationen *NodeAttr*, *EdgeAttr* sowie *LinkAttr* gespeichert. Damit lassen sich die Relationen in drei Gruppen einteilen:

1. Topologie
2. Daten-Tabellen
3. Typologie

Erstere beinhaltet alle Informationen darüber, welcher Knoten an welcher Kante wie (bezüglich Rolle und Richtung) teilnimmt. Die Topologie des Hypergraphen wird vollständig durch die Link-Tabelle abgebildet. Zur Gruppe der Daten-Tabellen zählen alle Relationen, die einzelne Entitäten und deren Eigenschaften vorhalten. Dazu gehören also Knoten und Kanten sowie die Attribute.

Zur Typologie zählen alle Tabellen, die das Typsystem für Knoten, Kanten, Rollen und Attribute in der Datenbank definieren. Tabelle 2 zeigt eine Zusammenfassung des Datenbestandes in Anlehnung an diese Einteilung. Da nur ein Bruchteil der Links in mehreren (zwei) Rollen auftreten, werden diese für die Betrachtungen dieser Arbeit vernachlässigt.

Links	729.407
Multi-Links	1.831
Knoten	257.462
Kanten	339.819
Link Attribute	747.776
Kanten Attribute	389.589
Knoten Typen	35
Kanten Typen	54
Link Typen (Rollen)	49

Tabelle 2: Anzahl und Typ von Knoten und Hyperkanten in WossiDiA. Mit Multi-Links sind dabei Links gemeint, die mehrere Rollen einnehmen. Die Daten belaufen sich, anders als in der vorangegangenen Tabelle, nur auf tatsächlich in der Hypergraphenstruktur vorkommenden Elemente.

Mit dieser Einteilung wird deutlich, dass Anfragen, die ausschließlich Hypergraphstrukturen auswerten, auf der Link-Tabelle arbeiten. Ein Beispiel ist die Frage, ob zwei Knoten über eine Kante verbunden sind. Kommen darüber hinaus noch Kriterien bezüglich der Knoten oder Kanten hinzu, müssen auch die Datentabellen ausgewertet werden. Die Typologie hingegen dient dazu, das Datenmodell in die Datenbank abzubilden, und ist daher nicht Gegenstand rechenintensiver Anfragen. Daher müssen vor allem die Topologie- und Daten-Tabellen für eine parallelisierte Verarbeitung aufbereitet werden.

Vor dem Hintergrund dieser ersten Ziele und Anforderungen können nun verschiedene technische Konzepte vorgestellt und auf ihre Relevanz und Übertragbarkeit auf den Anwendungsfall hin untersucht werden. Dazu stellt das folgende Kapitel zunächst die technischen Aspekte und Implikationen einer parallelisierten Anfrageverarbeitung im Kontext verteilter Datenbanksysteme vor. Diese paart das darauffolgende Kapitel 4 mit der Betrachtung und formalen Beschreibung von Graphen und Hypergraphen. So können die Eigenheiten und damit individuellen Vor- und Nachteile beider Ebenen in die Diskussion um einen Lösungsvorschlag einfließen. Beleuchtete man hingegen nur Datenbanktechniken, würden womöglich nützliche Eigenschaften der Hypergraphstruktur unterschlagen oder bestehende Techniken, die zur Problemlösung beitragen könnten,

nicht genutzt werden. Eine reine Fokussierung auf Graphtechniken ist umgekehrt ebenfalls nicht zielführend, da sich die hier gewonnenen Vorteile womöglich nicht im Rahmen eines VDBMS umsetzen ließen.

3 PARALLELE UND VERTEILTE DATENBANKSYSTEME

Parallelität kann dazu beitragen, die Verarbeitung großer Datenmengen zu beschleunigen. Daher bieten moderne Datenbanksysteme verschiedene Techniken auf der logischen und physischen Ebene an, um eine parallele Anfrageverarbeitung zu unterstützen. Auf der physischen Ebene können verschiedenste Architekturen zum Einsatz kommen. Bereits auf einer einzelnen Maschine können mehrere Threads oder Prozesse genutzt werden. Wird die Datenbank auf verschiedene Rechner aufgeteilt, handelt es sich um ein verteiltes Datenbanksystem (VDBMS). Für den Fall einer engeren Kopplung, beispielsweise durch gemeinsam genutzte Ressourcen oder der Anwendung in einem Cluster, spricht man eher von einem parallelen Datenbanksystem. Viele Konzepte und Eigenschaften treffen jedoch auf beide Ansätze zu. Für das WossiDiA-System ist eine Shared Nothing Architektur vorgesehen, bei der die Rechnerknoten ohne gemeinsam genutzte Ressourcen arbeiten und über ein Netzwerk kommunizieren. Daher sollen im Folgenden einige für die Anfrageparallelisierung wichtige Konzepte und Eigenschaften verteilter Datenbanksysteme beleuchtet werden.

Zunächst bringt ein VDBMS alle Vorzüge einer herkömmlichen Datenbank mit sich, von der Sicherung der Datenintegrität bis zur Pufferverwaltung beim Zugriff auf Primär- und Sekundärspeicher. Für den außenstehenden Nutzer bleibt die Verteilung der Daten auf mehrere Rechnerknoten transparent. Intern bietet das VDBMS die notwendigen Werkzeuge an, um Relationen auf die einzelnen Knoten aufzuteilen und so eine parallele Anfrageverarbeitung zu ermöglichen. Da es sich außerdem bei WossiDiA nicht um einen statischen Hypergraphen handelt, müssen Operationen wie Insert, Update und Delete unterstützt werden. Hier bietet ein VDBMS die nötige Transaktionssicherheit, sodass jeder Datenknoten stets einen konsistenten Zustand inne hält.

3.1 Datenverteilung

Hier unterscheidet man ganz allgemein die *logische Aufteilung* (teilweise auch *Fragmentierung* genannt) der Daten in verschiedene Segmente und deren *physischer Allokation* auf mehrere Knoten. Die wohl einfachste Form bildet eine Replikation, bei der keine logische Aufteilung vorgenommen und die gegebene Relation auf mehreren Knoten redundant gespeichert wird. Besonders für große Tabellen eignet es sich jedoch, eine Partitionierung vorzunehmen. Dabei wird zunächst eine logische Zerlegung vorgenommen. Die entstehenden Teile können

	Attr1	Attr2
T1	1	"A"
T2	2	"B"
T3	3	"C"
T4	4	"D"
T5	5	"E"

Abbildung 4: Eine zweidimensional partitionierte Relation mit fünf Tupeln und zwei Attributen. Dabei wurde zunächst vertikal und dann *Attr1* zusätzlich horizontal partitioniert. Die Partitionen sind durch Farben gekennzeichnet.

dann individuell auf die gegebenen Datenknoten verteilt werden.

Logische Einteilung (Fragmentierung)

Teilt man die Einträge der Tabelle horizontal auf, spricht man von einer *horizontalen Partitionierung*. Werden stattdessen die einzelnen Attribute auf verschiedenen Knoten gespeichert, handelt es sich um eine *vertikale Partitionierung*. Beide Ansätze können auch kombiniert werden, Abb. 4 gibt ein Beispiel. Im Allgemeinen lohnt sich eine horizontale Variante besonders bei Tabellen mit sehr vielen Tupeln, während eine vertikale Zerlegung bei Attributen mit besonders umfangreichem Inhalt wie z.B. Texten, auf die typischerweise nicht gemeinsam zugegriffen wird, attraktiv ist. Für das WossiDiA-System ist daher vornehmlich erstere relevant.

Partitionierungskriterium
Im Fall einer horizontalen Partitionierung muss ein Kriterium definiert werden, nach dem einzelne Tupel einer Partition zuzuordnen sind. Hier lassen sich z.B. eine zufällige Verteilung, ein Attributwert als Selektor oder Hash-Werte nutzen. Stehen zwei Relationen in einer inhaltlichen Beziehung zueinander, beispielsweise durch eine Schlüssel-Fremdschlüsselbeziehung, und für eine der beiden liegt bereits eine Zerlegung vor, kann diese genutzt werden, um eine Partitionierung für die andere abzuleiten. Nutzt man eine Kombination aus mehreren Kriterien, handelt es sich um eine *mehrdimensionale Verteilung*, andernfalls spricht man von einer *eindimensionalen Zerlegung*. Beispielsweise werden quadratische Matrizen oft in beiden Dimensionen partitioniert, um typische Aufgaben wie Matrix-Vektor oder Matrix-Matrix Multiplikationen gut zu unterstützen. Entscheidend für jedes verteilte Schema ist jedoch, dass die ursprünglichen Relationen stets rekonstruierbar sein müssen, damit globale Anfragen korrekt transformiert werden können.

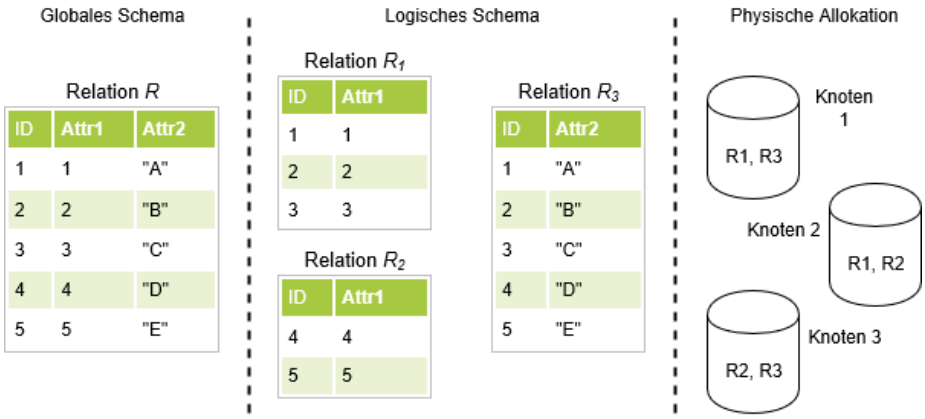


Abbildung 5: Ein möglicher Verteilungsentwurf für die bereits verwendete Beispielrelation R. Der hybride logischen Partitionierungsansatz wird physisch fortgeführt und eine Mischung aus redundanter und dedizierter Speicherung verwendet.

Allokation

Für eine logische Zerlegung ergeben sich verschiedene Möglichkeiten, die fragmentierten Relationen auf mehrere Knoten zu verteilen. Neben der bereits erwähnten Replikation mag man hier zunächst an die Strategie denken, jedem Knoten genau eine Partition zuzuweisen. Beide Herangehensweisen lassen sich aber wiederum auch gemischt verwenden, in dem beispielsweise eine Tabelle in mehrere Teile zerlegt, und jeder Teil auf mehrere Knoten repliziert wird. Eine andere Variante besteht darin, eine ausgewählte Partition einer horizontalen Zerlegung auf alle Knoten zu replizieren, während alle anderen Partitionen nur einem oder wenigen Knoten zugewiesen werden. In jedem Fall muss jedoch zusammen mit der physischen Verteilungsstrategie die Anzahl der Datenknoten festgelegt, sowie deren Ressourcenbedarf bedacht werden. Eine Übersicht über die Zusammenhänge zwischen globalem Schema, der logischen Partitionierung sowie der Allokation bietet Abb. 5.

Ein Verteilungsentwurf muss die durch den Anwendungsfall, die Datenstruktur und die zur Verfügung stehende Hardwareplattform gesetzten Anforderungen auf allen drei Ebenen einbeziehen, um eine gute Verteilung zu erreichen. Dabei beeinflussen sich diese zum Teil auch gegenseitig. Die zur Verfügung stehende Hardware limitiert die Anzahl sinnvoll parallel nutzbarer Rechnerknoten und deren mögliche Ressourcenverteilung. Auch der Umfang der gegebenen Daten hat hier Einfluss auf die Architektur. Werden zu wenig Rechnerknoten genutzt, sehen sich diese einer zu großen Belastung gegenüber und die gewünschte Skalierung wird nicht erreicht, werden jedoch zu viele genutzt, ist der Nutzen im

Vergleich zum zusätzlichen Aufwand zu gering. Außerdem muss eine Lastbalancierung gewährleistet werden, die typischerweise mit einer balancierten Datenverteilung einhergeht. Die Struktur der Daten spielt dabei ebenfalls eine wichtige Rolle, da logisch zusammenhängende Daten auch physisch gemeinsam abgelegt werden sollten. Eine Tabelle, die Vor- und Nachnamen speichert, sollte nicht vertikal partitioniert und auf zwei Knoten aufgeteilt werden, da typischerweise auf beide Attribute gemeinsam zugegriffen wird. In diesem Beispiel spiegelt sich auch der Zusammenhang mit den zu erwartenden Anfragen wieder, welche die zu entwerfende Strategie beeinflussen. Diese sollen daher im Folgenden näher betrachtet werden.

3.2 Parallelisierung von Anfragen

Zunächst werden Anfragen von einem Koordinator entgegengenommen, der mit den einzelnen Datenknoten kommuniziert und die Transaktionssicherheit gewährleistet. Dabei handelt es sich um Anfragen gegen das globale Schema, die interne Verteilung ist für den Nutzer transparent. Liegt eine Relation verteilt auf mehreren Knoten vor, muss das VDBMS eine gegebene Anfrage daher entsprechend der gewählten Verteilungsstrategie für die parallele Bearbeitung umformen. Dazu sind zwei Schritte notwendig:

1. Logische Umformung der Anfragen über dem globalen Schema zu äquivalenten Anfragen über dem internen, fragmentierten Schema

2. Verteilung und Optimierung der resultierenden Anfragen aus logischer und kostenbasierter Sicht

Jeder Datenknoten erhält dadurch eine auf ihn zugeschnittene Teilanfrage, deren Ergebnis am Ende zum Gesamtresultat beiträgt. Wie diese Umformung aussieht, hängt unter anderem von der Verteilungsstrategie ab. Liegt eine horizontale Zerlegung einer Relation vor, kann beispielsweise eine einfache Selektion auf allen Knoten parallel ausgeführt und dann die Ergebnismengen vereinigt werden. Dabei ist während der Vereinigung keine Duplikateliminierung notwendig, da kein Tupel in zwei verschiedenen Partitionen vorkommen kann. Handelt es sich hingegen um eine vollständig replizierte Relation, ist eine solche Parallelverarbeitung nicht ohne weitere semantische Informationen möglich, die Selektion würde auf allen Knoten zum selben Ergebnis führen.

Logische Transformation

Eine zentrale Aufgabe des VDBMS ist es also, aus einer Anfrage auf Relationen des globalen Schemas eine äquivalente Anfrage auf das logisch verteilte Schema abzuleiten und daraus anhand der gegebenen Allokation einen verteilten Anfrageplan zu erstellen. Zur Bewältigung der ersten Aufgabe gibt es mit der relationalen Algebra eine fundierte theoretische Grundlage, auf der die logische Transformation der Anfrage beruht. Diese ist rein logischer Natur und damit zunächst unabhängig von der zugrundeliegenden Allokation der Daten. Als Beispiel soll hier die zuvor beschriebene Selektion dienen. Für eine gegebene Relation R und deren logische (horizontale) Partitionierung in $R_1, ..., R_k$ kann eine Selektion über dem Prädikat P dank einiger Gesetzmäßigkeiten (u.A. die Distributivität der Selektion) wie folgt umgeformt werden: $\sigma_P(R) = \sigma_P(R_1 \bigcup R_2 \bigcup ... \bigcup R_k) = \bigcup_{i=1}^{k} \sigma_P(R_i)$. Ob so eine Umformung jedoch stattfindet, ist bereits Teil der Anfrageoptimierung, die bei verteilten Datenbanksystemen die physische Verteilung berücksichtigen muss. Beispielsweise ließe sich auch die ursprüngliche Relation aus den einzelnen Partitionen zusammensetzten, um dann die gewünschte Anfrage unverändert ausführen zu können. Da die Partitionen jedoch oft auf mehreren Knoten verteilt liegen und über ein Netzwerk mit hohem Zeitaufwand versandt werden müssten, ist eine Umformung in parallel ausführbare Teilanfragen in aller Regel vorzuziehen.

Allokation und Optimierung

Während logische Transformationen mittels festgelegter Regeln, die zu großen Teilen auch in den Anfrageoptimierern "normaler" Datenbanksysteme zum Einsatz kommen, erfolgt, stellt die physische Allokation der Relationen auf mehrere Rechnerknoten eine besondere Herausforderung für verteilte Datenbanksysteme dar. Zum einen muss das ACID Prinzip über alle Knoten hinweg zugesichert werden. Das bedingt ein verteiltes Transaktionsmanagement, das über entsprechende Protokolle abgewickelt werden muss. Da das Augenmerk beim WossiDiA-System jedoch auf der Optimierung von Lesetransaktionen liegt, ist ein anderer Punkt von zentraler Bedeutung. Statt eines "normalen" Ausführungsplans wird nun ein verteilter Plan benötigt, der die Zusammenarbeit aller betroffenen Datenknoten koordiniert.

Hier stellt sich die zentrale Frage, welcher Knoten welche Teilanfragen übernimmt und was dafür kommuniziert werden muss. Für den zuvor geschilderten Fall einer einfachen Selektion auf einer horizontal in zwei Partitionen R_1, R_2, die auf je einem Knoten vorliegen, zerlegten Relation R gestaltet sich dies noch rela-

tiv einfach. Die Anfrage kann anhand der oben beschriebenen Regel zerlegt, die Teilanfragen an ihren jeweiligen Knoten gesendet und das Ergebnis zusammengeführt werden. Daneben bestünde noch die (meist unattraktive) Möglichkeit, zunächst eine Relation vollständig auf den entsprechenden anderen Knoten zu übertragen, dort zusammenzufügen und dann die Anfrage unverändert abzuarbeiten.

Partition Pruning
Mehr Optionen eröffnen sich, wenn über das der Selektion zugrundeliegende Attribut mehr Informationen vorliegen. Wurde beispielsweise die Partitionierung aus selbigem Attribut abgeleitet, kann anhand des Partitionierungskriteriums festgestellt werden, in welcher der beiden Partitionen passende Tupel liegen können. Es kann sich dabei nur um eine Relation handeln, da aufgrund der Disjunktheit kein Attributwert in beiden Partitionen vorkommen darf. Dieses Vorgehen wird *Partition Elimination* oder auch *Partition Pruning*[2] genannt und ist Teil der Eliminierung irrelevanter Teilanfragen innerhalb der Anfrageoptimierung.

Constraint Exclusion
Dieser Ansatz lässt sich noch verallgemeinern und auf andere Kriterien neben dem Partitionierungskriterium anwenden. Dabei werden Partitionen anhand auf ihnen definierter Wertebereiche ausgeschlossen. Beispielsweise kann auf drei Partitionen R_1, R_2, R_3 ein Attribut $A2$ jeweils die Wertebereiche $1 - 60, 50 - 110$ und $100 - 160$ annehmen. Da nach diesem Attribut offenbar nicht partitioniert wurde, kann das Partition Pruning, wie es zuvor beschrieben wurde, hier nicht direkt zum Einsatz kommen. Sind diese Wertebereiche jedoch durch Constraints auf den jeweiligen Relationen festgelegt (für das gegebene Beispiel in Form entsprechender Check-Klauseln), lässt sich für eine Selektion über $A2$ mindestens eine, im besten Fall sogar zwei Partitionen ausschließen. Je nach Umfeld werden unter Partition Pruning beide Techniken zusammengefasst, bei Postgres findet sich für die allgemeinere Technik der Begriff *Constraint Exclusion*.

Bedeutung für Join-Operationen

Unabhängig davon, welche Variante für die Eliminierung irrelevanter Partitionen zum Einsatz kommt, gilt, dass diese essentiell für eine effiziente Anfrageverarbeitung ist. Besonders deutlich wird dies im Falle eines Join. Als Beispiel soll hier ein Self-Join der Tabelle R mit den numerischen Attributen $A1$ und $A2$

[2] Der Begriff Partition Pruning findet sich insbesondere im Oracle Umfeld

dienen, die abgeleitet von $A1$ horizontal in drei Teile zerlegt und wiederum auf drei Knoten verteilt wurde. Der Verbund soll über $A2$ gebildet werden. Ohne weitere Informationen über die Verteilung von $A2$ muss nun jede Partition mit sich selbst und jeder anderen verbunden werden, was in diesem Fall in sechs Joins resultiert[3]. Für alle Verbundoperationen zwischen verschiedenen Partitionen müssen die Knoten benötigte Daten austauschen. Ist jedoch wie zuvor für jede Partition ein Wertebereich vorgegeben, reduziert sich die Anzahl der nötigen Verbünde auf Partitionen, die eine Überlappung im Wertebereich aufweisen. Darüber hinaus können für die verbleibenden Joins zwischen verschiedenen Partitionen die Constraints der beteiligten Relationen per Konjunktion verknüpft und als Selektion vorangestellt werden. Damit lassen sich nicht nur irrelevante Partitionen sondern auch Tupel eliminieren. Ob sich dieser Schritt lohnt, ist wiederum Gegenstand der kostenbasierten Anfrageoptimierung.

Bedeutung für die kostenbasierte Anfrageoptimierung

Die Aufgabe einer *kostenbasierte Anfrageoptimierung* ist es, anhand von Kostenkriterien den Aufwand eines Ausführungsplanes zu schätzen und so einen möglichst günstigen zu finden. Als Kriterien dienen unter anderem die Größe einer Relation, die Selektivität eines Attributes oder die Existenz eines Idexes. Bei einem verteilten Datenbanksystem stellt dies eine nochmals schwierigere Herausforderung als bei herkömmlichen Datenbanken dar. Bereits für sequentielle Ausführungen auf einem Knoten ist der Suchraum über alle möglichen Ausführungspläne so groß, sodass oftmals Heuristiken zum Einsatz kommen, da eine vollständige Suche nicht durchführbar ist. Bei einer verteilten Berechnung ist zusätzlich das Zusammenspiel verschiedener Knoten zu koordinieren, wobei es Kommunikationskosten für Datenaustausch zwischen Knoten zu berücksichtigen gilt. Darüber hinaus erhöht sich durch die Zerlegung die Anzahl der Relationen. Im WossiDiA-System ist die Typisierung der Knoten über Vererbung implementiert, was bei einer Partitionierung der Knotentabelle in zehn Teile bei fünf Typen bereits zu fünfzig Tabellen führt. Ein schon für klassische Aufführungspläne exponentiell in der Anzahl an Joins wachsender Suchraum explodiert damit nochmals. Ob dies auch praktische Auswirkungen im konkreten Anwendungsfall hat, kann in dieser Arbeit jedoch nicht umfassend untersucht werden.

[3] Wiederum ausgenommen den Fall, dass zunächst die Ursprungsrelation rekonstruiert wird

Ziele des Verteilungsentwurfs

Wie bei herkömmlichen Datenbanksystemen ist es auch für ein VDBMS von großer Wichtigkeit, aufwendige Joins zu vermeiden. Hier kommt als Kostenfaktor jedoch die Datenübertragung zwischen einzelnen Knoten hinzu, die das System massiv ausbremsen kann. Deshalb ist bereits beim Verteilungsentwurf darauf zu achten, unnötige Kommunikation zwischen den Knoten schon durch Designentscheidungen zu vermeiden und gleichzeitig eine parallele Verarbeitung zu ermöglichen. Diese Ziele lassen sich nicht gleichermaßen uneingeschränkt erreichen sondern müssen gegeneinander abgewogen werden. Eine zufällige, gleichmäßige Verteilung würde zwar einen hohen Grad an Parallelität bieten, aber gleichzeitig zu enormem Kommunikationsaufwand führen. Umgekehrt führt eine Aufteilung der Daten in stark voneinander unabhängige Teile dazu, dass Anfragen, die vollständig auf einzelnen dieser Teile arbeiten, zwar ohne Kommunikationsaufwand jedoch auch ohne Parallelität verarbeitet werden.

3.3 Anfragen an das WossiDiA-System

Wie das verteilte Datenbanksystem eine oder mehrere Anfragen auf verschiedenen Knoten verteilt, um eine möglichst parallele Auswertung zu erzielen, hängt also sowohl vom Verteilungsentwurf als auch von der Art der Anfrage ab. Um einen hohen Grad an Parallelität zu gewinnen, müssen der Verteilungsentwurf und typische Anwendungsfälle gut aufeinander abgestimmt sein. Daher sollen für die zuvor in Abschnitt 2.3 motivierten Anforderungen und Aufgaben im Folgenden konkretisiert werden.

Zunächst ist festzuhalten, dass sich das WossiDiA-System überwiegend mit umfassenden Lesetransaktionen konfrontiert sehen wird. Schematisch bewegt sich damit der Anwendungsfall tendenziell im Bereich des Online Analytical Processing (OLAP). Im Gegensatz zu beispielsweise einem Data Warehouse sollen Änderungsoperationen jedoch nicht über eine Art ETL-Prozess integriert werden, sondern unmittelbar für den Nutzer möglich sein. Dennoch liegt das Hauptaugenmerk auf der Optimierung von ggf. langlaufenden Lesetransaktionen. Eine Durchsatzoptimierung ist nicht primäres Ziel einer Parallelisierung. Damit stellt sich die Frage, welche Relationen zentrales Ziel der Anfragen sind und wie diese verknüpft werden.

Zunächst lassen sich die Anfragen in zwei elementare Gruppen einordnen:

1. *Datenanfragen*

2. *Strukturanfragen*

Erstere operieren hauptsächlich auf der Knotenmenge, aber auch auf den Kanten- und Attribut-Tabellen. Strukturauswertungen finden hingegen auf der Link-Tabelle statt. Umfangreichere Anfragen kombinieren dann Teilanfragen aus beiden Bereichen.

Datenanfragen bedeuten vor allem Selektionen, Projektionen und Aggregationen auf der Knoten-Tabelle. Beispielsweise lassen sich Zeiten oder Orte auf metrische Eigenschaften wie Abstand oder Mittelpunkt hin untersuchen. Eine Anwendung stellt dabei die Bestimmung räumlicher oder zeitlicher Cluster dar. Für Anfragen, die Hypergraphstrukturen auswerten, stehen hingegen vor allem Selektionen und Self-Joins auf der Link-Tabelle im Fokus. Die Link-Tabelle speichert nur elementare Strukturinformationen direkt, komplexere lassen sich über Self-Joins konstruieren. Ein erster Join über das Kantenattribut liefert auf diese Weise alle Pfade der Länge eins und resultiert in Tupeln der Form $(Knoten, Kante, Knoten)$. Eine darauf aufbauende, typische Anfrage ist die nach der Nachbarschaft eines Knoten, die sich mit der gerade beschriebenen Strategie realisieren lässt. Der Codeabschnitt 1 zeigt eine solche Anfrage in SQL[4]. Um umfassendere Strukturen auszuwerten, werden weitere Self-Joins notwendig. Im Allgemeinen fallen für einen Pfad der Länge n genau $2n-1$ solcher Operationen an, da stets zwei Links mit gemeinsamer Kante einen einelementigen Pfad bilden.

```
SELECT DISTINCT l2.NodeID
FROM Link l1, Link l2
WHERE l1.EdgeID = l2.EdgeID
        AND l1.NodeID != l2.NodeID
        AND l1.NodeID = '123';
```

Codeausschnitt 1: SQL Anfrage, die die Nachbarschaft für den Knoten '123' liefert. Die erste Selektion stellt dabei die Verbundbedingung dar, die zweite eliminiert mögliche Schleifen und die dritte beschränkt die Pfade auf den gewünschten Knoten. Schlussendlich bildet die Projektion auf die Knoten-ID die übrigen Pfade auf die Nachbarschaft ab.

Umfangreichere Anfragen setzten inhaltsbasierte und struckturbasierte Informationen zusammen. Die dabei im Mittelpunkt stehende Operation ist der Natural Join von Knoten- und Linktabelle, die den Zusammenhang zwischen Daten und

[4] Der Lesbarkeit halber wurden Katalog-Präfixe entfernt sowie Attribute umbennant.

Struktur herstellt. Eine Beispielanfrage ist in im Codeausschnitt 2 gegeben, die für "Hans Schmidt" alle anderen Personen bestimmt, die durch eine Kante direkt mit Hans verbunden sind.

```
SELECT DISTINCT p2.firstname, p2.lastname
FROM Person p1,  Person p2, Link l1, Link l2
WHERE l1.EdgeID = l2.EdgeID
        AND l1.NodeID != l2.nodeID
        AND p2.id = l2.nodeID
        AND l1.NodeID = p1.id
        AND p1.firstname = 'Hans'
        AND p1.lastname = 'Schmidt';
```

Codeausschnitt 2: SQL Anfrage, die alle Personen aus der Nachbarschaft (bzgl. der Hypergraphstruktur) von "Hans Schmidt" liefert.

3.4 Kriterien für einen Verteilungsentwurf

Um komplexe Anfragen effizient auf mehrere Knoten aufzuteilen, gilt es verschiedene Kriterien zu beachten:

- Geringer Kommunikationsaufwand für zu erwartende Anfragen
- Balancierung der Datenlast entsprechend zu erwartender Anfragen
- Hohe Parallelisierbarkeit zu erwartender Anfragen
- Änderungsoperationen (insbesondere das Einfügen neuer Kanten) sind zu unterstützen

Diese Kriterien beziehen sich dabei ausdrücklich auf typische Anfragen, da sich für jede Verteilung Anfragen finden lassen, die für diese besonders ungünstig bezüglich der genannten Kriterien sind. Für WossiDiA sollen insbesondere Anfragen, die lokale Hypergraphstrukturen wie Nachbarschaften und Konnektivität auswerten, sollen effizient unterstützt werden.

Das erste Kriterium wurde bereits in Abschnitt 3.3 als Ziel formuliert und anhand der Join-Operationen begründet. Die balancierte Verteilung der Daten ist ebenfalls entscheidend. Andernfalls kann es bei der Anfragebearbeitung zu Leistungseinbußen kommen, da weniger stark belastete Knoten ihre Rechenkapazität nicht ausschöpfen können und ggf. sogar auf stärker belastete Knoten warten müssen. Dies würde auch das dritte Kriterium verletzen, das eine zeitlich

parallele Abarbeitung typischer Anfragen fordert, um kürzere Antwortzeiten zu erzielen.

Darüber hinaus sollte eine Partitionierung robust gegenüber Änderungen am Datenbestand sein, d.h. kleine Änderungen sollten nicht dazu führen, dass die Daten vollständig neu partitioniert werden müssen, da dies mit nicht unerheblichem Kommunikations- und Rechenaufwand verbunden ist. Auch wenn es sich im Falle von WossiDiA um Archivdaten handelt, muss mit Änderungen am Graphen gerechnet werden. Diese tauchen sowohl im Zusammenhang mit neuen Archivinhalten als auch mit Änderungen wie z.B. Korrekturen am Bestand auf. Außerdem soll es möglich sein, neu gewonnene Erkenntnisse und Zusammenhänge in die Datenbank einzufügen.

Diese Ziele gleichermaßen zu erreichen, birgt einige Herausforderungen und ist nicht immer möglich, stattdessen muss zwischen den Anforderungen abgewogen werden. Beispielsweise kann man, statt perfekter Balance zwischen allen Knoten einhalten zu müssen, kleine Abweichungen bis zu einem bestimmten Wert in der Lastverteilung zulassen. Damit öffnet sich mehr Spielraum, um Daten bezüglich des Kommunikationsaufwandes besser zu verteilen. Außerdem eignet sich eine Partitionierung nicht für jede Art von Anfragen gleichermaßen und muss daher auf diese abgestimmt werden.

3.5 Ansätze für einen Verteilungsentwurf

Mithilfe dieser Erkenntnissen lassen sich bereits einige grundsätzlich sinnvolle Strategien für den WossiDiA-Hypergraphen ableiten. Zusammen mit den Vorbetrachtungen in Kapitel 2 ist festzuhalten, dass eine Verteilung der Link- und Knotentabelle im Fokus des Verteilungsentwurfes stehen muss. Für die Fragmentierung scheidet hier eine vertikale Partitionierung aus. Im Fall der Linktabelle wäre eine Aufteilung der Strukturinformationen kontraproduktiv und auf die Knotentabelle muss nur zugegriffen werden, wenn neben den Strukturinformationen die durch den Knoten repräsentierten Werte Ziel einer Anfrage sind. In beiden Fällen würde eine vertikale Zerlegung stark zusammenhängende Informationen trennen, auf die häufig gemeinsam zugegriffen wird. Darüber hinaus kann die gewünschte Skalierbarkeit so nicht erreicht werden.

Daher ist also eine horizontale Zerlegung vorzuziehen. Eine solche muss für Daten- und Struktur-Tabellen abgestimmt vorgenommen werden, sodass Knoten bzw. Kanten und deren anhängige Links später auf gemeinsame Datenstatio-

nen verteilt werden können. Andernfalls könnte die elementare Join Operation zwischen den beiden Tabellen, die einen Grundbestandteil typischer Anfragen darstellt, nur mit erheblichem Kommunikationsaufwand und damit nicht effizient durchgeführt werden. Selbiges gilt für die Attribut-Relationen von Knoten, Kanten und Links.

Die zweite elementare Join Operation stellt der Self-Join der Linktabelle mit sich selbst über die Kanten- sowie Knoten-ID dar. Damit eine horizontale Fragmentierung der Relation nicht zu massiven Kommunikationskosten führt, muss sie so vorgenommen werden, dass zwei Partitionen möglichst wenig Tupel mit gemeinsamen Kanten- und Knoten-IDs enthalten. Im besten Fall muss für den Join ausschließlich eine Partition mit sich selbst verbunden werden, nämlich dann, wenn die betroffenen Knoten- bzw. Kanten-IDs nur in einer einzigen Partition vorkommen. Damit stellen sich diese zwei Attribute bereits als wichtige Elemente für ein Kriterium heraus, nach dem die Partitionierung vorgenommen wird.

Andere Strategien lassen sich hier ausschließen. Neben einer offenbar nicht zielführenden zufälligen oder Round-Robin Verteilung bestünde auch die Möglichkeit, Links entsprechend ihrer Typinformationen zu gruppieren. Da jedoch Hypergraphstrukturen auch unabhängig von ihrer Typisierung und Richtung ausgewertet werden sollen, bedeutete dies Leistungseinbußen bei entsprechenden Anfragen. Darüber hinaus ist eine aus der Typologie abgeleitete Fragmentierung im Hinblick auf Balancierung und Skalierbarkeit kritisch zu betrachten. Eine Zunahme an Daten lässt sich hier nicht einfach durch mehr Partitionen abdecken und für eine balancierte Datenverteilung muss der Datenbestand uniform über Typen oder Gruppen von Typen verteilt sein. Eine solche Verteilung erschwert außerdem den Umgang mit neu eingefügten oder gelöschten Typen.

Auf diese Weise zeichnet sich ein Verteilungsentwurf ab, der die Topologie horizontal partitioniert, auf mehrere Datenknoten verteilt und für die Datenrelationen daraus eine Verteilung ableitet. Außerdem steht mit den Knoten- und Kanten-IDs der Links die Hypergraphstruktur im Zentrum der Partitionierungsstrategie. Wie dieses Krieterium genau aussieht, wie viele Partitionen daraus gebildet und diese genau verteilt werden sollten, um für die beschriebenen Anfragen möglichst gut geeignet zu sein, bedarf jedoch weiterer Untersuchungen. Während die bisherigen Betrachtungen sich am gegeben konzeptionellen Schema und dessen Eigenschaften orientierten, stand das den Daten zugrundeliegende Hypergraphmodell noch nicht im Vordergrund des Verteilungsentwurfes. Es sind jedoch vor allem die Eigenheiten der zu verteilenden Daten, die für

ein Verteilungskriterium entscheidend sind. Daher sollen im folgendem Kapitel Hypergraphen und darauf aufbauende Techniken zur Partitionierung vorgestellt werden.

4 HYPERGRAPHEN UND GRAPHPARTITIONIERUNG

Graphen sind ein aus der Mathematik bekanntes Werkzeug, um Zusammenhänge zwischen Objekten zu modellieren und zu analysieren. Sie sind in der Theorie gut verstanden und kommen bei zahlreichen Anwendungsfällen in den verschiedensten Variationen zum Einsatz. Hypergraphen stellen dabei eine generalisierte Variante von Graphen dar. Im Gegensatz zu Graphen variieren hier jedoch die Definitionen in der Literatur, auch für den "einfachen" Fall eines ungerichteten, einfachen Hypergraphen [GLP93; Bre13; AL17]. Im folgenden sollen Graphen, Hypergraphen sowie einige für die Partitionierung notwendige Begriffe erklärt und formal eingeführt werden.

4.1 Graphen und Hypergraphen

Graphen

Ein Graph $G = (V, E)$ besteht aus einer Menge V von *Knoten* (engl. vertices) sowie einer Menge von *Kanten* $E \subseteq V \times V$ (engl. edges) zwischen den Knoten. Im Falle eines *ungerichteten* Graphen handelt es sich bei einer Kante um ein ungeordnetes Knotenpaar. Bei einem *gerichteten* Graphen (auch Digraph genannt) hingegen sind die Kanten geordnete Paare.
Graphen eignen sich besonders, um durch die Kanten Beziehungen zwischen Elementen explizit darzustellen. Allerdings beschränken sich diese auf jeweils paarweise Beziehungen. Hypergraphen erweitern die Definition einer Kante, um beliebig viele Knoten in einer Kante zuzulassen und so multilaterale Beziehungen zwischen mehreren Knoten modellieren zu können.

Hypergraphen

Der Definition von Claude Berge [Ber89] folgend verstehen wir unter einem ungerichteten *Hypergraphen* H eine Familie $(e_1, ..., e_m)$ von nichtleeren Teilmengen einer Knotenmenge V. Ein Beispiel eines solchen Hypergraphen ist in Abb. 6 grafisch dargestellt. Ein ungerichteter Graph ist ein Spezialfall eines Hypergraphen, nämlich der, in dem alle Kanten höchstens zwei Knoten beinhalten.

Wie bei Graphen kann man auch bei Hypergraphen eine zusätzliche Semantik einführen, um eine Richtung auszudrücken [GLP93; AL17]. Dabei wird eine Hyperkante als Tupel (T, H) definiert, wobei T die Menge der eingehenden und H die Menge der ausgehenden Knoten darstellt. Die zugehörige *Inzidenzmatrix*

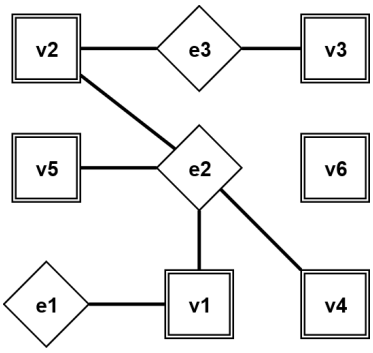


Abbildung 6: Ein Hypergraph mit 3 Kanten 6 Knoten. Rauten modellieren dabei Kanten, Quadrate stellen die Knoten dar. Die Zugehörigkeit eines Knotens zu einer Kante wird durch die Verbinder dargestellt.

wird für einen Hypergraphen mit n Knoten und m Kanten als $n \cdot m$ Matrix $A = [a_{ij}]$ gebildet, wobei:

$$a_{ij} = \begin{cases} 1, & v_i \in T(e_j) \\ -1, & v_i \in H(e_j) \\ 0, & v_i \notin e_j \end{cases}$$

In der Literatur variieren hier die Details der Definition bezüglich leerem Head bzw. Tail, Knoten die gleichzeitig sowohl eingehend als auch ausgehend sind oder Beschränkungen bezüglich der Kardinalität von H und T. So erlaubt Gallo beispielsweise keine gemeinsamen Knoten in Head und Tail, d.H. $H \bigcap T = \emptyset$ [GLP93]. Im Hinblick auf den Anwendungsfall folgt diese Arbeit der Definition in [MSH17], die eine Hyperkante $e = (H, T)$ mit $H, T \subseteq V$, wobei die leere Kante ausgeschlossen bleibt, ohne weitere Einschränkungen definiert. Für H und T schreiben wir auch $H(e)$ bzw. $T(e)$. Zur Vereinfachung wird außerdem für $v \in H(e) \bigcup T(e)$ auch $v \in e$ verwendet, um auszudrücken, dass ein Knoten v Teil der Kante e ist, unabhängig von der Richtung. Die weiteren Definitionen leiten sich aus denen normaler Graphen ab und folgen dabei im wesentlichen Gallo [GLP93].

Knotengrad
Mit $deg(v) = |\{e \in E \; | v \in e\}|$ bezeichnen wir den *Grad* des Knoten v, also die Anzahl aller an v hängenden Kanten. Umgekehrt verstehen wir unter dem *Kantengrad* die Kardinalität $|e|$ einer Kante e, also die Anzahl aller in e enthaltenen Knoten. Weiterhin nennen wir die Menge aller Kanten, in denen v im Tail teilnimmt, den *Forward Star* von v: $FS(v) = \{e \in E, v \in T(e)\}$. Analog

dazu ist der *Backward Star* durch $BS(v) = \{e \in E, v \in H(e)\}$ gegeben.
Mit diesen und weiteren, darauf aufbauenden Werkzeugen lässt sich analysieren, wie ein Knoten im Graphen vernetzt ist.

Pfade und Konnektivität
Neben den soeben geschilderten Eigenschaften einzelner Knoten ist auch die Frage, welche Knoten wie über Kanten zusammenhängen, von großer Bedeutung, die daher im folgenden formalisiert werden soll.
Ein *Pfad* ist eine alternierende Sequenz $(v_1, e_1, ..., e_n, v_{n+1})$ von Knoten und Kanten, sodass für alle $i \in \{1, ..., n\}$ gilt: $v_i, v_{i+1} \in e_i$. Ein *gerichteter Pfad* verlangt dagegen, dass $v_i \in T(e_i)$ und $v_{i+1} \in H(e_i)$ gilt und berücksichtigt damit die Richtung, in denen die Knoten an Kanten beteiligt sind.
Dieser Pfad heißt *einfach*, falls keine Kante mehrfach vorkommt und *elementar*, wenn kein Knoten mehrfach vorkommt. Existiert zwischen zwei Knoten v, w ein solcher Pfad, sind diese verbunden.
Die Anzahl aller in einem Pfad vorkommenden Kanten stellt die Länge des Pfads dar. Die *Distanz* bezeichnet die Länge des kürzesten Pfades zwischen zwei Knoten.

Nachbarschaft, k-Nachbarschaft
Als Nachbarschaft eines Knoten v wird die Menge aller Knoten bezeichnet, mit denen v unmittelbar eine Kante teilt. Dieses Konzept lässt sich von der unmittelbaren Umgebung auf einen Abstand von bis zu k Kanten erweitern. Bei einer gegebenen natürlichen Zahl $k \geq 1$ und einem Knoten v spricht man bei der Menge aller über einen ungerichteten Pfad der Länge $\leq k$ zu v verbundenen Knoten von der k-Nachbarschaft.

Cut
Mit der Semantik des Cuts soll im Weiteren eine wichtige Formalisierung einer Aufteilung innerhalb eines Hypergraphens eingeführt werden, die für Partitionierungen eine wichtige Rolle spielt.
Unter einem *Cut* verstehen wir die Einteilung der Knoten V in zwei nichtleere, disjunkte Mengen S und $V \setminus S$ und schreiben dafür $\kappa = (S, V \setminus S)$. Man beachte, dass jede nichtleere Teilmenge der Knoten also auf natürliche Weise einen Cut bildet. Per Konvention gibt man daher für einen Cut auch einfach die kleinere Teilmenge an.
Die Menge aller Kanten, die Knoten aus beiden Teilen des Cuts enthalten, und damit Verbindungen zwischen den Knotenmengen darstellen, wird als *Cut-Set* bezeichnet: $set(\kappa) = \{e \in E \mid v, w \in e,\ v \in S,\ w \in V \setminus S\}$.
Die Anzahl aller Kanten, die Knoten aus beiden Teilen des Cuts enthalten, nen-

nen wir *Cut-Size*: $size(\kappa) = |set(\kappa)| = |\{e \in E \mid v, w \in e, v \in S, w \in V \setminus S\}|$. Man beachte, dass, in Abweichung zu manchen Definitionen in der Literatur [GLP93], hier v und w nicht über die Kante verbunden sein müssen, sondern auch beide im Head oder beide im Tail vorkommen dürfen.

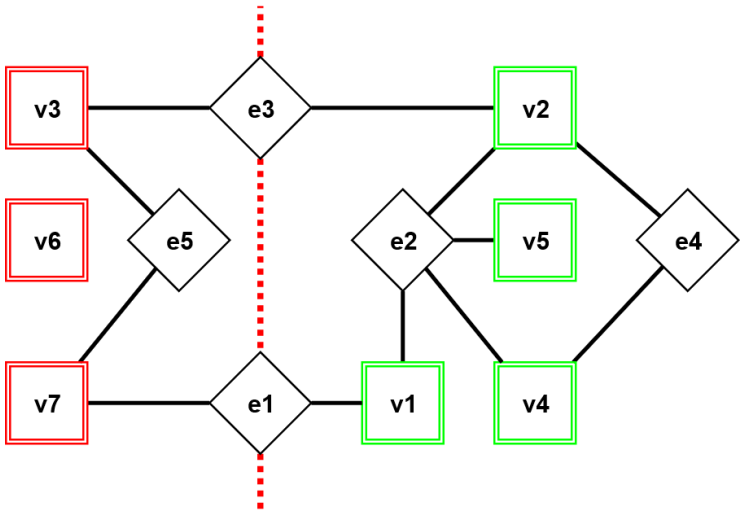


Abbildung 7: Ein ungerichteter Hypergraph mit 7 Knoten und 5 Kanten. Ein Cut, der durch die gestrichelte Linie angedeutet wird, teilt die Knotenmenge so auf, dass $e2, e4$ und $e5$ interne Kanten und $e1$ sowie $e3$ geteilte Kanten darstellen. Die Knoten sind entsprechend ihrer zugehörigen Teilmenge farbig markiert.

Abb. 7 zeigt einen solchen Cut mit hervorgehobenem Cut-Set in einem Hypergraphen. Durch einen Cut lassen sich zwei Klassen von Kanten unterscheiden: Interne, die ausschließlich Knoten innerhalb einer Teilmenge vereinen, und externe, die Knoten aus beiden Teilen des Cuts beinhalten, d.h. die Teil des Cut-Sets sind. Das Cut-Set im dargestellten Graphen besteht aus $\{e1, e3\}$, während $\{e2, e4, e5\}$ jeweils vollständig innerhalb einer Teilmenge liegen. Daraus ergibt sich eine Cut-Size von zwei.

Eine weitere, nützliche Unterscheidung kann bei dem zuvor eingeführten Knotengrad getroffen werden. Für einen Cut $\kappa = (S, V \setminus S)$ bezeichnen wir für einen Knoten v mit

$$deg_{int}(v) = |\{e \in E | v \in e, e \notin set(\kappa)\}|$$

den internen Knotengrad und mit

$$deg_{ext}(v) = |\{e \in E | v \in e, e \in set(\kappa)\}|$$

den externen Knotengrad. Damit unterscheiden wir für einen Knoten zwischen den an ihm hängenden internen und externen Kanten. In der Summe ergeben beide zusammen den Knotengrad: $deg(v) = deg_{int}(v) + deg_{ext}(v)$.
Im Beispielgraphen aus Abb. 7 ergibt sich so für den gemeinsamen Knoten der Kanten $e1$ und $e2$ ein interner Grad $deg_{int}(v) = 1$ und ein externer Grad $deg_{ext}(v) = 1$.

Induzierter Subgraph
Mithilfe der soeben beschriebenen Semantik eines Cut lassen sich leicht weitere Eigenschaften beschreiben. So nennt man einen Hypergraphen zusammenhängend, wenn es keinen möglichen Cut mit leerem Cut-Set gibt. Gibt es jedoch solche Cuts, nennt man die Teilmengen Komponenten. Intuitiv bedeutet das, dass Komponenten aus der Perspektive der Kantenstruktur vollkommen unabhängig voneinander sind.
Auch wenn der Hypergraph zusammenhängt, ist es gelegentlich von Vorteil, einen bestimmten Teil isoliert vom Rest des Graphen zu betrachten. Für eine Teilmenge $S \subset V$ ist $H_S = (S, E_S)$ mit $E_S = \{e \in E \mid T(e), H(e) \subseteq S\}$ der von S induzierte Subgraph auf $H = (V, E)$. Anschaulich heißt das, dass der Subgraph nur solche Kanten enthält, deren Knoten alle in S liegen.

Mithilfe dieser Formalismen lässt sich das Problem der Partitionierung nun beschreiben, umsetzten und bewerten.

4.2 Partitionierung

Im Allgemeinen versteht man unter einer Partitionierung die vollständige Aufteilung einer Menge auf eine festgelegte Anzahl nichtleerer Teilmengen. Formal bedeutet das für eine Menge V und eine Zerlegung $\Pi = (V_1, ..., V_k)$ in k Partitionen $V_i \subseteq M, i \in \{1, ..., k\}$, dass

$$\begin{aligned} &1) \quad V_i \neq \emptyset \\ &2) \quad \bigcup_{i=1}^{k} V_i = V \\ &3) \quad V_i \bigcap V_j = \emptyset,\ i \neq j \end{aligned}$$

Für Graphen und Hypergraphen besteht das klassische Problem der Partitionierung darin, eine solche für die Menge der Knoten geeignet zu wählen und dabei bestimmte Kriterien zu beachten. Hier sind Ziele von besonderer Bedeutung, die auch für den WossiDiA Hypergraphen relevant sind. Zum einen sollen alle Partitionen möglichst gleich groß und zum anderen das entstehende Cut-Set möglichst klein sein. Diese bereits in Abschnitt 2.3 motivierten Ansprüche sollen

ebenfalls kurz formalisiert werden.

Eine balancierte Partitionierung (k, ν) ist eine Zerlegung $\Pi = (V_1, ..., V_k)$, sodass $max_i\ |V_i| \leq \nu\lceil\frac{|V|}{k}\rceil$. Jede Teilmenge beinhaltet damit höchstens $\nu \cdot (|V|/k)$ Elemente. Für $\nu = 1$ spricht man von einer uniformen oder auch perfekten Partitionierung.

Das Konzept des Cut-Sets lässt sich von einer Bisektion auf eine Zerlegung Π in k Teilmengen übertragen: $set(\Pi) = \bigcup_{i=1}^{k} set(V_i)$. Für die Cut-Size wird oftmals eine leicht veränderte, unter anderem als $(\lambda - 1)$ und *Communication Volume* bekannte Metrik[5] anstelle von $|set(\Pi)|$ genutzt [Cat+07; TK08; Dev+06]. Diese wird für einen Hypergraphen $H = (V, E)$ definiert als

$$comVol(\Pi) = \sum_{e \in E} \lambda(e, \Pi) - 1$$

wobei $\lambda(e, \Pi)$ die Anzahl der Partitionen darstellt, die mindestens einen Knoten in e haben. Der Unterschied zur zunächst intuitiven Kardinalität des Cut-Sets besteht darin, dass Kanten, die gleichzeitig mehrere Partitionen verbinden, so entsprechend der Anzahl der Teilmengen, die sie verbinden, gezählt werden, anstatt nur einmalig.

In den folgenden Abschnitten werden sowohl Graphen als auch Hypergraphen beleuchtet. Um Unklarheiten zu vermeiden, werden daher im Weiteren Kanten des Hypergraphen stets ausdrücklich als *Hyperkanten* bezeichnet.

Partitionierung der Knoten von (Hyper-) Graphen

Das Problem einer balancierten Partitionierung lässt sich damit wie folgt formulieren: Für einen gegebenen Hypergraphen H ist für einen festgelegten Balancierungsfaktor ν sowie ein festes k eine balancierte Partitionierung (k, ν) mit der kleinstmöglichen Cut-Size gesucht.
Dabei handelt es sich bereits für eine Aufteilung eines klassischen Graphen in zwei uniforme Partitionen um ein NP vollständiges Problem, das auch Minimum Bisection Problem genannt wird. Andrev und Räcke zeigen sogar, dass eine $(k, 1)$ balancierte Zerlegung nicht bis auf einen endlichen Faktor approximiert werden kann, es sei denn $P = NP$ [AR04]. Wagner et. al. stellen fest, dass die Problemstellung mit zunehmend strikter werdenden Balancierung schwieriger wird [WW93].
Da Graphen spezielle Hypergraphen darstellen, gilt dies auch für Hypergraphen.

[5] Je nach Anwendungsgebiet finden sich die Begriffe *Communication Volume* und *Fanout*

Darüber hinaus sind etliche andere ähnliche oder mit einer Partitionierung verbundene Probleme auf Graphen und Hypergraphen ebenfalls NP-vollständig [ŠS06]. Um der damit verbundenen Komplexität der Aufgabe zu begegnen, werden oftmals heuristische Verfahren eingesetzt.

Partitionierung der Kanten von (Hyper-) Graphen

Neben der klassischen Graphpartitionierung besteht auch die Möglichkeit, Kanten anstatt von Knoten zu partitionieren. Smorodkina et. al. führen den Begriff Balanced Edge Partitioning ein und zeigen, dass sich die Komplexität des Problems ähnlich zu dem der Knotenpartitionierung verhält und ebenfalls NP-vollständig sowie nicht bis auf einen endlichen Faktor approximierbar ist (es sei denn $P = NP$) [STT07]. Allerdings definieren die Autoren keine Partitionierung der Kanten im eigentlichen Sinne, sondern eine Zerlegung der Knoten, bei der Kanten balanciert werden. Spätere Arbeiten setzten eine Balancierung der Kanten im Rahmen einer tatsächlichen Aufteilung der Kanten um [BLV14]. Dabei wird, äquivalent zum Cut-Set über Kanten, ein Cut-Set über der Knotenmenge definiert. Hängt ein Knoten an mehreren, aus unterschiedlichen Partitionen stammenden Kanten, gehört dieser zum Cut-Set. Die Cut-Size ergibt sich wiederum aus der Größe des Cut-Sets, alternativ kann hier auch die bereits für Hypergraphen eingeführte $(\lambda - 1)$ Metrik zum Einsatz kommen, da Knoten an mehrere, in verschiedenen Partitionen liegende Kanten angrenzen können.

Dieser Ansatz ist sowohl aus theoretischer als auch aus praktischer Sicht weniger erforscht als die klassische Variante, kommt jedoch insbesondere bei aktuellen graphbasierten Systemen zur Lastverteilung zum Einsatz [Gon+12; Hei+16]. Dabei werden weniger Replikationen von Knoten sowie bessere Lokalität und damit insgesamt weniger Kommunikationsaufwand als Gründe angeführt. Das gilt besonders für Graphen, die Sachverhalte aus der "echten Welt" abbilden, wie zum Beispiel soziale Netzwerke oder Wissensrepräsentationen, da diese oft Eigenschaften wie die sogenannte Small-World-Property[6] und eine reziproke Exponentialverteilung des Knotengrades aufweisen. Hier hat sich gezeigt, dass eine Zerlegung nach Knoten mit klassischen Algorithmen zu schlechten Ergebnissen führen kann [Lan04; Les+08; ARK06]. Eine Partitionierung der Kanten resultiert in deutlich besseren Ergebnissen [Che+14; Che+15; Li+17; Zha+17a].

Der Ansatz einer Kantenpartitionierung lässt sich von klassischen Graphen leicht auf Hypergraphen übertragen, indem Hyperkanten partitioniert werden.

[6] Man spricht von einem Small-World-Network, wenn die durchschnittliche Distanz zwischen zwei Knoten klein, typischerweise proportional zu $log(|V|)$, ist

Die Zielfunktion lässt sich dabei eins zu eins übernehmen. Einen ersten direkten Ansatz verfolgen Yang et. al. und stellen mit *HyperSwap* einen Algorithmus vor [Yan+16]. Frühere Arbeiten bzw. Systeme nutzten dazu entweder eine zufällige (Round-Robin, Hash) Verteilung oder greifen auf einen Graphen als Repräsentation zurück. Letztere Option stellt eine Möglichkeit dar, mit Hypergraphen umzugehen, und bedarf daher näherer Aufmerksamkeit.

4.3 Zusammenhang zwischen Graphen und Hypergraphen

Wie zuvor erwähnt, stellen Hypergraphen eine Generalisierung von Graphen dar. Es ist jedoch möglich, Hypergraphen durch Graphen darzustellen, was sich in zwei Varianten umsetzten lässt:

- *Star-Expansion*
- *Clique-Expansion*

Abb. 8 illustriert einen Hypergraphen zusammen mit beiden Darstellungsformen als Graph.

Bei der *Star-Expansion* wird ein gegebener Hypergraph als bipartiter Graph dargestellt. Ein bipartiter Graph ist dabei ein Graph, der durch einen Cut so geteilt werden kann, dass Kanten ausschließlich zwischen den beiden Partitionen verlaufen. Ein solcher kann einen Hypergraphen $H = (V, E)$ wie folgt abbilden:

1. Die Knotenmenge V bleibt erhalten und wird für jede Hyperkante um einen Knoten ergänzt
2. Die Teilnahme eines Knotens an einer Hyperkante wird dann durch eine Kante vom Knoten zu dem die entsprechende Hyperkante repräsentierenden, neu eingefügten Knoten modelliert

Auf diese Weise stellt eine Partition die eigentliche Knotenmenge und die andere die ursprüngliche Hyperkantenmenge dar. Diese Transformation ist aus praktischer Sicht besonders angenehm, da die Inzidenzmatrix des Hypergraphen direkt als Adjazenzmatrix des bipartiten Graphen übernommen werden kann und nur noch aufgefüllt werden muss.

Eine zweite Option ist die sogenannte *Clique-Expansion*. Dabei werden wiederum die Knoten übernommen, die Hyperkanten jedoch durch direkte Kanten modelliert. Dazu werden alle in einer Hyperkante enthaltenen Knoten paarweise miteinander verbunden. Anders als in der Darstellung als bipartiter Graph

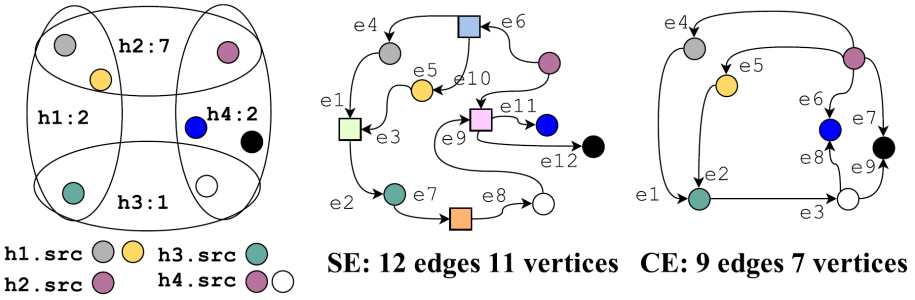


Abbildung 8: **Links:** Gerichteter und gewichteter Hypergraph. Der Knotes des Head sind dabei in der Notation <*Hyperkante*>.*src* aufgelistet. **Mitte:** Die Darstellung in der Star-Expansion. **Rechts:** Im Vergleich dazu die Clique-Expansion. Quelle [HZY15]

werden hier Hyperkanten nicht mehr direkt modelliert. Da zwei Knoten durch mehrere Hyperkanten direkt verbunden sein können, muss die Information, wie viele Hyperkanten eine Kante der Clique-Expansion abbildet, als Annoation oder Kantengewicht gespiechert werden. Soll eine Hyperkante rekonstruierbar sein, muss sogar die Menge der ursprünglichen Hyperkanten an jede Kante des neuen Graphen annotiert werden.

Bedeutung der Graphrepräsentationen für das Partitionierungsproblem

Beide Verfahren haben individuelle Vor- und Nachteile und eignen sich nicht für jede Zielstellung gleichermaßen. Als wichtigen Anwendungsfall ist hier die Repräsentation eines Hypergraphen innerhalb eines Systems wie z.B. GraphX [Xin+14] zu nennen. Eine Diskussion im Rahmen von parallelisierten Algorithmen für maschinelles Lernen auf Hypergraph zu den Auswirkungen einer solchen verteilten Speicherung führt [HZY15]. Einen empirischen Vergleich beider Strategien für selbigen Anwendungsfall findet sich in [Hei+16]. Wenngleich eine Speicherung als Graph für das WossiDiA-System nicht genutzt werden soll, lassen sich aus den Ergebnissen Erkenntnisse für den vorliegenden Anwendungsfall ableiten.

Darüber hinaus besteht die Möglichkeit, anstatt den Hypergraphen direkt zu partitionieren, zunächst eine der beiden Transformationen zu einem Graphen vorzunehmen, für diesen dann eine Verteilung zu bestimmen und diese dann auf den ursprünglichen Hypergraphen zu übertragen. Je nach Art der Partitionierung und Zielfunktion lässt sich dieses Vorgehen nutzen.

Nutzung der Clique-Expnasion
Beispielsweise kann eine balancierte Partitionierung der Knoten bestimmt werden, indem die Repräsentation als Graph durch Clique-Expansion gebildet und dann mit einem Verfahren für Graphen partitioniert wird. Die Lösung kann als Lösung für den Hypergraphen interpretiert werden, in dem die Verteilung der Knoten eins zu eins übernommen wird. Dabei bleibt die Balancierung erhalten, die Zielfunktion ändert sich jedoch. Eine Minimierung des Cut-Sets des Graphen resultiert in der Minimierung der Anzahl aller einelementigen Pfade zwischen den Partitionen des Hypergraphen[7].

Nutzung der Star-Expnasion
Für die Darstellung als bipartiter Graph zeichnet sich ein anderes Bild. Würden hier schlicht alle Knoten partitioniert, ließe sich die Lösung nicht auf den Hypergraphen übertragen, da auch Knoten, die Hyperkanten repräsentieren, verteilt würden. Ignoriert man diese schlicht, kann eine Balancierung für die übrig bleibenden Knoten nicht mehr gewährleistet werden. Außerdem bedeutet ein Durchteilen der Kanten des bipartiten eine Teilung der Links des Hypergraphen und ist daher nicht zielführend.

Neben den semantischen Unterschieden zwischen Hypergraphen und deren Darstellungsformen als Graphen ist auch eine Differenz in der Problemgröße festzustellen. Ein Hypergraph $H = (V, E)$ wächst in der Darstellung als bipartiter Graph in der Knotenmenge zu $|V|+|E|$ und in der Kantenmenge auf mindestens[8] $2 \cdot |E|$. Für den Spezialfall, dass in einem Hypergraphen alle Hyperkanten genau zwei Knoten verbinden, verdoppeln sich damit Mengen. Eine Clique-Expansion hingegen würde für diesen Fall zu keinen neuen Kanten oder Knoten führen. Bei Hypergraphen mit sehr großen Hyperkanten entstehen jedoch sehr viele Kanten, da die Clique-Expansion quadratisch in der Größe einer Hyperkante neue Kanten bildet, während die Star-Expansion hier linear bleibt. Sowohl bei einer Nutzung als Darstellungsform als auch als Zwischenform sind diese Auswirkungen zu berücksichtigen und anhand der Eigenschaften des vorliegenden Hypergraphen zu bewerten. Dies gilt sowohl für Algorithmen als auch für Eigenschaften bezüglich der Balancierung, die für eine Lastverteilung besonders relevant sind und daher ebenfalls näher beleuchtet werden sollen.

[7] Vorausgesetzt, dass mehrere Hyperkanten zwischen zwei Knoten in der Clique-Expansion durch entsprechende Kantengewichte modelliert werden
[8] Für den typischen Fall, dass alle Hyperkanten mindestens zwei Knoten verbinden.

Balance einer Partitionierung von Graphen und Hypergraphen

Eine wichtige Unterscheidung zwischen Graphen und Hypergraphen stellt sich bei der Balancierung heraus. Die Problemstellung der Partitionierung fordert die Gleichverteilung nur für die Menge, die aufgeteilt wird. Die folgenden Abschnitte analysieren die daraus resultierenden Worst-Cases für die jeweils andere Menge.

Knotenpartitionierung eines Graphen
Bei der klassischen Problem der Graphpartitionierung werden Knoten, jedoch nicht Kanten balanciert. Hier lässt sich für einen zusammenhängenden Graphen festhalten, dass ein mögliches Ungleichgewicht der Kanten linear in der Anzahl der Knoten sowie Partitionen beschränkt ist. Für einen zusammenhängenden Graphen $G = (V, E)$ mit $|V| = n$ und einer balancierten Bisektion in $(S \subset V, V \setminus S)$ mit $|S| = k = \lceil \frac{n}{2} \rceil$ lässt sich der schlechteste Fall betrachten. Dieser liegt vor, wenn der von S induzierte Subgraph minimal zusammenhängend und der von $V \setminus S$ induzierte Subgraph vollständig ist. Damit finden sich für den kleineren Subgraphen genau $k - 1$ Kanten, für den größeren hingegen $\binom{(n-k)}{2} = \frac{(n-k)\cdot(n-k-1)}{2}$, was sich unter Vernachlässigung der Rundung von k im Fall einer ungeraden Anzahl von Knoten zu $\frac{k\cdot(k-1)}{2}$ vereinfachen lässt. Daraus ergibt sich ein größtmögliches Ungleichgewicht von $k/2 \approx n/4$, das hier als Verhältnis vom größtmöglichen zum kleinstmöglichen Wert gebildet wird.

Knotenpartitionierung eines Hypergraphen
Für Hypergraphen stellen sich wesentlich schlechtere Grenzen dar. Hier genügt bei selbiger Bisektion bereits eine einzige Hyperkante, um den kleineren Teil (bezüglich der Hyperkanten) des Cuts zu verbinden. Dem gegenüber beläuft sich die Anzahl aller möglichen Hyperkanten eines Subgraphen mit k Knoten[9] auf 2^k, was für einen Hypergraphen der Größe n eine mögliche Imbalance von bis zu $2^{n/2}$ bedeutet. Im Gegensatz zu Graphen ist diese also exponentiell statt linear in der Anzahl der Knoten und Partitionen beschränkt. Man bedenke dabei jedoch, dass es sich in beiden Fällen nur um die Mengen der vollständig innerhalb einer Partition liegenden Hyperkanten handelt, die des Cut-Sets sind nicht Teil dieser Betrachtung. Weiterhin bezieht sich die Argumentation tatsächlich nur auf zusammenhängende Graphen und Hpyergraphen, im allgemeineren Fall stellen sich schlechtere Worst-Case Fälle heraus.

Kantenpartitionierung eines Graphen

[9] Streng genommen $2^k - 1$, da die leere Hyperkante nicht erlaubt ist. Für die hier diskutierte Approximierung wird dies jedoch vernachlässigt

Auch der umgekehrte Fall lässt sich aus theoretischer Sicht betrachten, bei dem die Menge der Kanten balanciert partitioniert wird. Für Graphen lässt sich die vorherige Überlegung umkehren. So können k Kanten höchstens $k+1$ Knoten zusammenhängend verbinden. Die minimale Anzahl ergibt sich mit $\min_n$ $k \leq n \cdot (n-1)/2$, was sich in etwa mit $\sqrt{k}$ abschätzen lässt und damit einen Faktor von $\frac{k+1}{\sqrt{k}} \approx \sqrt{k}$ liefert.

Hyperkantenpartitionierung
Auch hier stellt sich für Hypergraphen eine schlechtere Grenze heraus. Während bei Graphen die Anzahl der Knoten in einer Kante auf zwei beschränkt ist, kann eine Hyperkante beliebig viele Knoten enthalten. Damit ist die Anzahl der Knoten in einer Partition weder durch Hyperkanten noch Partitionen sondern nur in der Knotenmenge beschränkt. Eine Partition mit k Hyperkanten muss jedoch mindestens $\lfloor log_2(k) \rfloor \approx \lfloor log_2(n/2) \rfloor = \lfloor log_2(n) \rfloor - 1$ Knoten enthalten. Daraus lässt sich eine Imbalance von grob $\frac{n}{log_2(n)}$ ableiten.

Diese Grenzen gilt es zu bedenken, wenn herkömmliche Verfahren zur Bestimmung einer Partitionierung angewendet werden, die nur eine der beiden Mengen balancieren. Davon soll eine Auswahl im Folgenden kurz vorgestellt werden.

4.4 Algorithmen zur Graphpartitionierung

Es gibt im wesentlichen zwei Herangehensweisen, um Graphen mit mehr als einer handvoll Knoten zu partitionieren:

- Lokale, meist heuristische Verfahren
- Globale Verfahren

Lokale Ansätze nehmen an einer Anfangspartitionierung schrittweise punktuelle Verbesserungen vor. Globale Herangehensweisen setzt mit einer Betrachtung des Graphen als Ganzes an. Viele Verfahren verbinden beide Ansätze, um vor allem für große Graphen gute Lösungen zu finden. Im Folgenden sollen einige Ansätze kurz vorgestellt und bezüglich des Anwendungsfalls auf ihre Anwendbarkeit hin bewertet werden. Eine umfassendere Übersicht für Graphen findet sich in [Bul+13], speziell mit Hypergraphen befasst sich [PM07].

Zum einen gibt es exakte Algorithmen, die durch geschicktes Einschränken des Suchraums für kleine (Hyper-) Graphen die optimale Lösung bestimmen. Für den Hypergraphen des WossiDiA-Systems können aufgrund der Größe jedoch nur heuristische Verfahren zum Einsatz kommen. Ein erster Algorithmus wurde

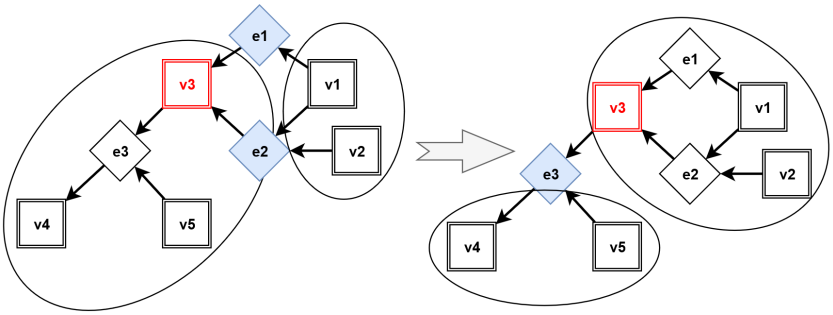


Abbildung 9: Ein Hypergraph mit 3 Hyperkanten 5 Knoten. Die Ellipsen deuten die Aufteilung der Knoten an. Der rot markierte Knoten $v3$ wird im dargestellten Verbesserungsschritt in die andere Partition verschoben, da dies die größte Verkleinerung des Cut-Sets bedeutet.

von Keringhan und Lin vorgestellt [KL70], den Fiduccia und Mattheyses später auf Hypergraphen sowie um einen Balancierungsfaktor erweiterten [FM88]. Dabei wird eine Greedy-Strategie verfolgt, die durch lokale Optimierungen die optimale Lösung anstrebt. Die Grundidee besteht darin, eine initiale Partitionierung durch verschieben eines Knotens oder vertauschen zweier Knoten zu verbessern. Dieses Vorgehen ist anhand eines Beispielhypergraphen in Abb. 9 illustriert. In jedem Schritt werden alle zu Hyperkanten des Cut-Sets inzidenten Knoten betrachtet und bestimmt, ob und wie sehr sich das Cut-Set bei einer Übernahme in eine andere Partition verkleinert (sofern die Balancierungsforderung dies erlaubt). Der Knoten, der die größte Verbesserung bedeutet, wird dann gewählt und das Vorgehen wiederholt, solange, bis keine Optimierung mehr erzielt werden kann. Dieser Ansatz eignet sich besonders, wenn eine gute, initiale Partitionierung auf natürliche Weise vorgegeben oder leicht zu bestimmen ist. Beispielsweise setzte Facebook eine modifizierte Variante dieser Methode für dessen Social Graph ein, da sich hier eine gute initiale Partitionierung aus der geografischen Verteilung der Nutzer ableiten lässt [Pre14].

Ein anderer Ansatz stammt aus der linearen Algebra und nutzt die Dualität zwischen Graphen und Matrizen. Dabei wird zunächst die Laplace-Matrix $L = D - A$ aus der Adjazenzmatrix A und der Degree-Matrix D gebildet, wobei D die Diagonalmatrix mit den Knotengraden darstellt. Für L wird dann der zweitkleinste Eigenwert und der zugehörige Eigenvektor bestimmt, aus dem sich eine Bisektion ableiten lässt [Fie73]. Weitere Partitionen erhält man durch rekursives Anwenden der Methode, die auch als *Spectral Bisection* bekannt ist.

Die wohl erfolgreichsten Verfahren sind jedoch sogenannte *Multilevel-Verfahren*. Die Idee besteht darin, einen Graphen zunächst durch Zusammenfassen von Knoten und Kanten zu abstrahieren und zu verkleinern, für den einfacheren Fall eine Lösung zu bestimmen und diese dann auf den ursprünglichen Graphen anzuwenden. Diese drei Phasen werden oft mit *Coarsening – Partitioning – Refinement* bezeichnet. Besonders für große Graphen lohnt es sich dabei, Abstraktions- und Refinement-Schritte mehrfach anzuwenden, bevor eine Partitionierung bestimmt wird. In der zweiten Phase, d.h. auf der höchsten Ebene der Abstraktion, ist eine initiale Zerlegung zu bestimmen und dann zu verbessern. Hier kann eines der bereits vorgestellten Verfahren zum Einsatz kommen. Nach jedem Refinement-Schritt können dann wiederum lokale Heuristiken eingesetzt werden, um die Qualität der Lösung zu verbessern. Dieses Vorgehen eignet sich besonders für große Graphen und erfreut sich auch dank der unterschiedlichen Kombinationsmöglichkeiten mit verschiedenen Heuristiken und Coarsening-Methoden großer Verbreitung. Daher wurde der Ansatz auch auf Hypergraphen erweitert [KK99].

Darüber hinaus gibt es weitere Verfahren und Heuristiken, wie z.B. probabilistische Ansätze [Kab+17], aber auch geometrische Methoden, die sich auf Problemstellungen anwenden lassen, bei denen die Knoten des Graphen eine geometrische Bedeutung haben. Andere Varianten betrachten eine Streaming-Version des Problems, bei dem der Graph nicht vollständig vorliegt sondern die Daten verteilt über einen Zeitraum eintreffen und ad-hoc partitioniert werden müssen [AIV15]. Sowohl für Graphen als auch Hypergraphen gibt es einige frei verfügbare Softwarepakete, die entsprechende Algorithmen implementieren, z.B. die METIS Familie [KK95], Zoltan [Dev+02; Dev+06] oder PaToH [CA01; Cat+07].

Zusammenhänge zwischen Partitionierung und Clustering
Abschließend ist noch zu bemerken, dass die Problemstellung der Graphpartitionierung eng mit dem Problem, Cluster in Graphen zu finden, verwandt ist. Gemeinsames Ziel beider Ansätze ist es, möglichst voneinander isolierte Teilbereiche zu bestimmen und entsprechende Verfahren können durchaus zu ähnlichen Lösungen führen. Im Gegensatz zur Partitionierung wird beim Clustering jedoch in der Regel keine Balancierung gefordert und oftmals die Anzahl an Teilmengen nicht vorgegeben. Außerdem unterscheiden sich die Zielfunktionen subtil voneinander: Clusterverfahren optimieren die Distanz von Knoten zum Mittelpunkt eines Clusters anhand einer Distanzmetrik während eine Partitionierung die Größe des Cut-Sets berücksichtigt. Daher ist zwischen beiden Problemstellungen zu differenzieren. Dennoch sind Clusterverfahren für das Partitionierungspro-

blem von Bedeutung und können beispielsweise in der Coarsening-Phase von Multilevel-Verfahren zum Einsatz kommen oder dazu genutzt werden, eine initiale Partitionierung zu bestimmen.

5 ÜBERTRAGUNG UND ANWENDUNG

Mit Hilfe der Datenbanktechinken aus Kapitel 3 sowie den formalen Werkzeugen und Betrachtungen für Graphen und Hypergraphen aus Abschnitt 4 sollen nun verschiedene Optionen für das WossiDiA-System vorgestellt und diskutiert werden. Diese orientieren sich an konkreten Fragestellungen, die sich aus den vorangegangenen Betrachtungen ergeben. Im Mittelpunkt dieser Fragen steht dabei die Verbindung graphentheoretischer Erkenntnisse mit Datenbanktechniken unter Berücksichtigung der spezifischen Anforderungen und Eigenheiten des Anwendungsfalls. Für einen Verteilungsentwurf und dessen Umsetzung leiten sich die folgenden Fragen ab:

1) *Über welche Dimensionen wird die Zerlegung der Hypergraphenstruktur gebildet? Neben Knoten und Hyperkanten stehen hier noch die Richtung sowie das Typsystem mit Knoten-, Hyperkanten- und Link-Typ zur Verfügung. Welche sind davon auszuwählen und ggf. zu kombinieren?*

2) *Wie werden die gewählten Partitionen auf einzelne Rechnerknoten verteilt?*

3) *Welche Eigenschaft soll optimiert werden?*

4) *Über welche Dimensionen werden Forderungen bezüglich der Balancierung gestellt?*

5) *Mit welchen Verfahren lässt sich eine solche Verteilung gewinnen?*

6) *Wie viele Partitionen werden gebildet?*

Dabei beeinflussen sich manche Designentscheidungen gegenseitig. So eignet sich nicht jeder Algorithmus für jede Art der Zerlegung oder Optimierung der gewünschten Zielfunktion. Dennoch ist die Bandbreite an Möglichkeiten so groß, dass eine Diskussion aller Optionen den Rahmen dieser Arbeit sprengen würde, von einem empirischen Vergleich mehrerer Umsetzungen ganz zu schweigen. Ziel der Arbeit ist vielmehr, von existierenden Ansätzen auszugehen, deren Übertragbarkeit zu untersuchen und so Lösungen herauszuarbeiten. Den Fragen 1) bis 4) gehen die Kapitel 5.1 bis 5.4 auf den Grund. Auf welche Weise sich bestimmte Strategien umsetzten lassen, wird dabei zusammen mit dem jeweiligen Ansatz angerissen. Welche Algorithmen für eine Umsetzung am besten geeignet sind, kann diese Arbeit jedoch nicht umfassend beantworten, da diese Frage, sowie die nach einer geeigneten Anzahl der Partitionen, vor allem aus praktischer Sicht zu beantworten ist. Eine Umsetzung einer k-Wege-Zerlegung erfolgt in dieser Arbeit daher für mehrere Werte von k.

Bezug zu bestehenden Umsetzungen

Bei der Betrachtung existierender Techniken zur Partitionierung von Hypergraphen ist zunächst festzuhalten, dass diese in den meisten Fällen dazu dienen, anwendungsspezifische Probleme zu lösen, die als Partitionierungsproblem auf einem Hypergraphen formuliert werden. Damit unterscheidet sich das Ziel von einer Verteilung um der parallelen Verarbeitung des Hypergraphen selbst willen. Dadurch ergeben sich unterschiedliche spezifische Anforderungen. Modelliert ein Hypergraph beispielsweise durch Knoten individuelle Rechenaufgaben und durch Hyperkanten notwendige Kommunikation zwischen diesen, kann durch eine Partitionierung die Menge der Rechenaufgaben auf verschiedene Prozessoren verteilt werden. Dabei ist der Kommunikationsaufwand zwischen den Prozessoren zu minimieren, die Kommunikation innerhalb eines Prozessors ist jedoch nicht von Interesse. Während hier eine balancierte Verteilung der Hyperkanten nicht von wesentlicher Bedeutung ist, spielt diese für eine parallele Auswertung des Hypergraphen durchaus eine Rolle. Letztere betrachten die bereits erwähnten Arbeiten [HZY15; Hei+16], die sich jedoch in zwei wesentlichen Punkten von den Anforderungen dieser Arbeit abgrenzen lassen.

1. Beiden Ansätzen mit Spark bzw. GraphX auf Frameworks für verteilte Berechnungen auf, während dem zu entwickelnden Ansatz eine verteilte Datenbank zugrunde liegt.

2. Die Arbeiten befassen sich mit der Ausführung von Algorithmen im Bereich des maschinellen Lernens, die viele Änderungensoperationen, besonders auch an Knotenwerten, durchführen. Daraus erwächst die Notwenigkeit, Replikationen von Knoten möglichst zu vermeiden, da diese zu Performanzverlusten führen. Für die überwiegend aus Leseoperationen bestehenden Anfragen an das WossiDiA-System gilt dies jedoch nicht

Diese Unterschiede sind bei der Betrachtung und Übertragung existierender Ansätze zu berücksichtigen.

5.1 Art der Partitionierung

Eine Zerlegung der Hypergraphenstruktur bedeutet eine Partitionierung der Link-Tabelle. Als wichtige Dimensionen wurden in Abschnitt 3.4 bereits Knoten und Hyperkanten herausgestellt, die sich in drei Varianten kombinieren lassen.

1. *Edge-Cut:* Eindimensionale Partitionierung über der Knoten-ID

2. *Vertex-Cut:* Eindimensionale Partitionierung über der Kanten-ID

3. *Hybrid-Cut:* Zweidimensionale Partitionierung über Knoten- und Hyperkanten-ID

Die Bezeichnungen orientieren sich an [Hei+16]. Der *Edge-Cut* bedeutet aus graphtechnischer Sicht ein Zerlegung der Knotenmenge. Damit kommt jede Knoten-ID in genau einer Partition der Link-Tabelle vor, eine Kanten-ID kann jedoch in mehreren oder sogar allen Partitionen vorkommen. Beim *Vertex-Cut* tritt das umgekehrte Szenario ein. Abb. 10 stellt beide Varianten schematisch dar. Man beachte, dass die angedeuteten Zerlegungen keine Links durchtrennen, da dies die Disjunktheit verletzten würde. Ein *Hybrid-Cut* kombiniert beide Ansätze. Abb. 11 illustriert ein Beispiel, in dem drei Partitionen gebildet wurden. Anders als bei beiden vorangegangenen Strategien handelt es sich dabei um eine zweidimensionale Partitionierung über Knoten- und Kanten-IDs.

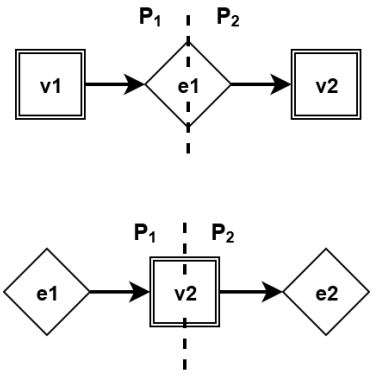


Abbildung 10: **Oben:** Ein durch Edge-Cut in zwei Partitionen geteilter Hypergraph. **Unten:** Dem entgegen ein Vertex-Cut.

Die eindimensionale Zerlegung im Falle des Vertex- und Edge-Cuts bedeutet für die elementaren Join-Operationen über dem partitionierten Attribut, dass diese vollständig lokal und damit ohne Kommunikationsaufwand parallel auf den Datenknoten ausgeführt werden können. Der Self-Join reduziert sich zu einzelnen Self-Joins der Partitionen mit sich selbst, der Join mit der entsprechenden Datentabelle reduziert sich ebenfalls auf Joins zwischen den einander entsprechenden Partitionen. Verbundoperationen auf dem jeweils anderen Attribut erfordern hingegen Datenaustausch über das Netzwerk, wobei im schlechtesten Fall alle Partitionen mit sich selbst und allen anderen zu verbinden sind. Ein Edge-Cut führt zu schnellen Berechnungen des Knotengrades sowie des Forward- und Backward-Stars durch den Verbund der Link-Relation über das Knotenattribut während ein Vertex-Cut eine effiziente Berechnung der Nachbarschaft von Knoten begünstigt, da hier der Join über die Kanten-ID gebildet wird.

Im Unterschied zu Edge- und Vertex-Cut ist für den Hybrid-Cut zunächst festzuhalten, dass hier für keines der beiden Attribute die striktere Semantik der Partitionierung alleine gilt. Darauf aufbauende Optimierungen wie beispielsweise Partition Pruning (im engeren Sinne) beim Zugriff auf ein einzelnes Attribut nicht mehr möglich sind. Die zuvor vorgestellten elementaren Operationen arbeiten jedoch stets auf einem der beiden Attribute, was ein Hybrid-Cut nicht ausnutzt. Dem sind zwei wesentliche Vorteile entgegen zu halten. Da nicht mehr nur Knoten oder nur Hyperkanten geteilt werden dürfen, stehen viel mehr Möglichkeiten für eine Zerlegung offen, was zu deutlich besseren Lösungen bezüglich Balancierung und Cut-Size führen kann. So lässt sich beispielsweise der Hypergraph in Abb. 11 sowohl durch einen Vertex-, Edge- als auch Hybrid-Cut in verschiedenster Weise zerlegen. Der zweite zu nennende Vorteil besteht darin, dass der Aufwand für die Strukturanfragen besser ausgeglichen wird. Während die eindimensionalen Strategien entweder Stern- oder Nachbarschaftsoperationen bevorzugen und die jeweils andere vollständig mit den Schnittstellen der Graphstruktur belasten, priorisiert ein Hybrid-Cut keine der beiden. Im Extremfall kann ein Hybrid-Cut natürlich auch nur Hyperkanten oder nur Knoten teilen, sofern das genutzte Verfahren nicht explizit für eine Balancierung beider Teile im Cut-Set optimiert. Solche Extreme ausgenommen, ist der Hybrid-Cut attraktiv, solange keine Operation bevorzugt werden soll.

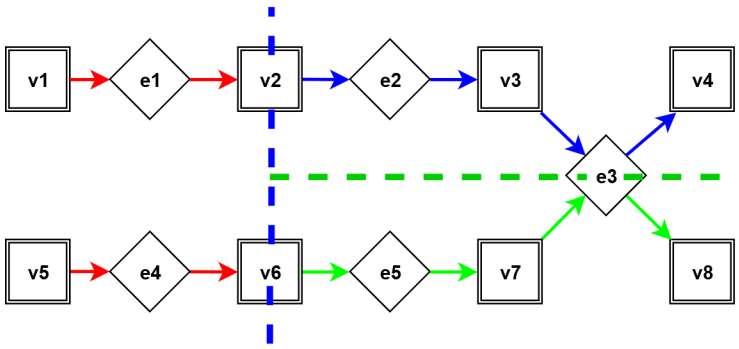


Abbildung 11: Ein durch einen Hybrid-Cut in drei Partitionen geteilter Hypergraph. Die Links sind entsprechend ihrer Partition farbig markiert.

Einfluss von Richtung und Typen

Wie bereits in Abschnitt 3.4 festgestellt, ist eine reine Verteilung nach Richtung oder Typ der Links nicht zielführend. Denkbar wäre jedoch eine Kombination zusammen mit einer der vorangegangen Varianten, beispielsweise ließe sich

nach Knoten und Knotentyp partitionieren. Allerdings führt eine Zerlegung nach Knoten-, Hyperkanten- oder Linktyp zu schlechter Lokalität, d.h. lokale Graphstrukturen würden zu sehr über verschiedene Partitionen zersplittert. Das liegt unter anderem daran, dass Orte, Personen, Zeiten und Begriffe auf verschiedene Weisen und oft auch direkt zusammenhängen. Beispielsweise verbindet eine eine Aktivität darstellende Hyperkante typischerweise Ort, Person, Zeit und Tätigkeit. Eine solche Zerlegung wird erst mit speziellen Anfragen, die verschiedene Zusammenhänge anhand ihres Typs isoliert voneinander betrachten, sinnvoll. Als Vorteil ist die geringere Anzahl an Relationen zu nennen, die dadurch entsteht, dass der Knotentyp durch Vererbung auf der Knotentabelle realisiert ist und sich damit weniger neue Relationen ergeben. Die Richtung eines Links lässt sich als Dimension ebenfalls ausschließen, da diese, je nach Typ des Links, unterschiedliche Bedeutungen haben können und Analysen auch unabhängig von der Richtung durchgeführt werden sollen.

5.2 Allokation

Unabhängig davon, ob ein Vertex-, Edge- oder Hybrid-Cut vorgenommen wird, ist die Semantik der Fragmentierung eines VDBMS zu beachten. Soll hier ein Teil der Knoten- oder Kanten-Tabelle repliziert werden, ist dazu eine eigene Partition zu bilden. Im Kontext anderer paralleler oder verteilter Systemen wird davon gesprochen, eine Menge zu partitionieren und die andere an den Schnittstellen zu replizieren, da hier logische und physische Aufteilung nicht explizit unterschieden werden müssen. Eine häufig verwendete Variante besteht darin, im Fall eines Vertex-Cuts die Knoten, durch die geteilt wird, auf alle Rechnerknoten zu replizieren, auf denen entsprechende Partitionen vorliegen. Äquivalent lässt sich dies für Hyperkanten umsetzten.

Innerhalb eines relationalen Datenbanksystems lässt sich diese Strategie jedoch nicht eins zu eins umsetzten, da dazu jeweils eine Partition für jede benötigte Kombination aus Datenknoten benötigt würde, was exponentiell in der Anzahl der Datenknoten steigt. Man beachte hierbei, dass dies bei einem Hypergraphen für alle Cut-Varianten zutrifft. Beispielsweise würden für eine Zerlegung in A, B, C und drei Datenknoten zusätzlich vier Partitionen hinzukommen, die Abb. 12 als Venn-Diagramm visualisiert. Damit ist zwar gewährleistet, dass jeder Knoten des Hypergraphen nur auf Datenknoten gespeichert wird, die ihn benötigen, die gestiegene Anzahl an Relationen ist jedoch nicht vertretbar. Selbst wenn die Zerlegung so gewählt wird, dass nicht zwischen allen Teilen Verbindungen im Cut-Set existieren, können diese nachträglich eingefügt werden, womit

die entsprechenden Partitionen zwingend notwendig werden würden. Darüber hinaus ist der Gewinn im dargestellten Beispiel gering, da ohnehin drei Viertel aller Hyperkanten des Cut-Sets auf alle Datenknoten abgelegt werden.

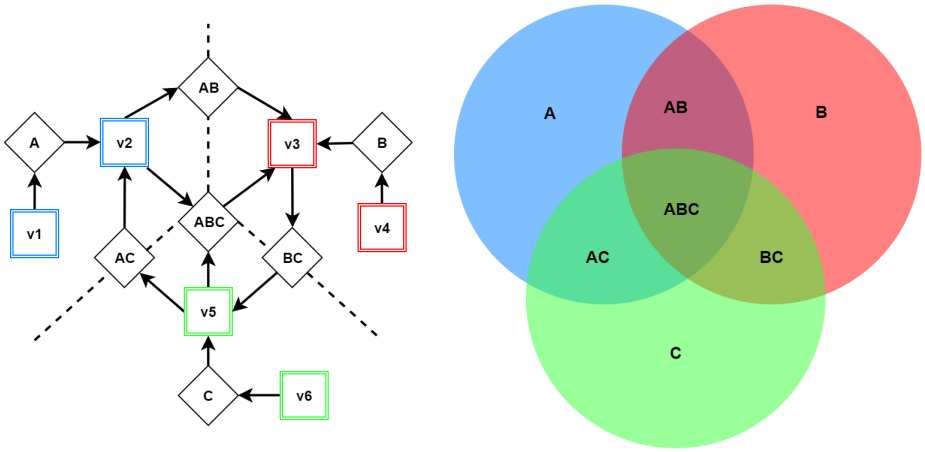


Abbildung 12: **Links**: Ein durch einen 3-Way-Edge-Cut partitionierter Hypergraph. Die Knoten sind durch Farben den Partitionen zugeordnet. Jede Partition wird einem Datenknoten zugewiesen. Die Hyperkanten tragen den Namen der Teilmengen, die sie verbinden. **Rechts**: Die entsprechende Aufteilung der Hyperkanten als Venn-Diagramm, wobei jeder Teil des Diagramms eine Partition darstellt. Darüber hinaus ist dies als Allokation zu interpretieren, wobei alle in einem Kreis liegenden Teilmengen dem Datenknoten des Kreises zugewiesen werden.

Eine Alternative besteht darin, eine Partition für die Knoten (respektive Hyperkanten) des Cut-Sets zu bilden und auf alle Datenknoten zu replizieren. Das bedeutet zwar potenziell einen hohen Grad an unnötiger Replikation, da Datensätze auf alle Datenknoten gespeichert werden, unabhängig davon, ob sie dort tatsächlich gebraucht werden. Die Anzahl der Relation bleibt dafür aber überschaubar und die Datenknoten halten alle Tupel lokal vor, die sie entsprechend der gegebenen Zerlegung der Topologie benötigen und müssen diese nicht über das Netzwerk anfordern. Für den Fall des Hybrid-Cuts lässt sich dies umsetzten, indem das Cut-Set des Vertex- und Edge-Cuts jeweils einer Partition zugewiesen und auf allen Datenknoten gespeichert wird.

Dieses Konzept lässt sich auch auf die eigentliche Zerlegung der Topologie ausweiten. Die Idee ist dabei, ausgehend von k gegebenen Partitionen, eine neue Partition zu schaffen, die ausschließlich gemeinsame Links der gegebenen Zerlegung enthält. Ein Beispiel, wie dieser Vorgang aussieht, ist in Abb. 13 illustriert. Die alten Partitionen bleiben, abzüglich der ins neue Teil gewanderten Links,

erhalten, was dazu führt, dass sie nur noch mit der einen, neuen Partition gemeinsame Links haben. Wird diese auf alle Datenknoten repliziert, kommen beide elementaren Strukturabfragen mit maximal zwei Joins aus und können vollständig lokal bearbeitet werden.

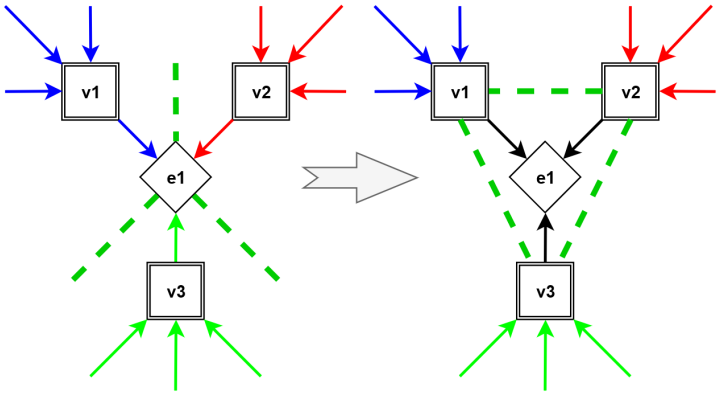


Abbildung 13: Transformation einer 3-Wege Knoten-Partitionierung zu einer 4-Wege Hyperkanten-Partitionierung. Die nicht verbundenen Pfeile deuten Links zu weiteren, nicht dargestellten Hyperkanten an. Vormals gemeinsame Links bilden nach der Transformation die neue Partition.

Neben dem offensichtlichen Vorteil, Anzahl und Aufwand der Verbundoperationen massiv zu reduzieren, hat dieses Vorgehen jedoch auch Nachteile. Zum einen wird ein vorheriger Vertex-Cut zum Edge-Cut und umgekehrt, ein Hybrid-Cut hingegen bleibt in der Art erhalten. Soll also ein Edge-Cut mit dieser Strategie vorgenommen werden, ist zunächst ein Vertex-Cut zu bestimmen. Steht dafür kein geeignetes Verfahren zur Verfügung, das zu guten Ergebnissen führt, sinkt die Qualität (Balancierungsgrad und Cut-Size) der Partitionierung. Darüber hinaus entstehen wiederum unnötige Replikationen, wobei nicht garantiert ist, dass diese gleichmäßig über alle Partitionen verteilt sind.

Je nach dem, welche Strategie verfolgt wird, ergeben sich unterschiedliche Schwerpunkte für die Zielfunktion der Partitionierung, auf die im Folgenden näher eingegangen werden soll.

5.3 Zielfunktion einer Partitionierung

Jede Partitionierungsstrategie benötigt eine zu optimierende Funktion. Um eine effiziente Verarbeitung von Anfragen zu gewährleisten, muss eine solche Umfang und Anzahl von Joins über verteilte Partitionen möglichst gering halten. Dabei

geht es zunächst um Self-Joins über der Link-Tabelle, aus der für die Datentabellen eine Zerlegung so abgeleitet wird, dass Joins zwischen Daten- und Linktabelle (hauptsächlich die Verbindung von Knoten- und Linktabelle) effizient durchgeführt werden können. Eine Zielfunktion für die Zerlegung des Hypergraphen sollte also zu wenig Links mit gemeinsamen Kanten- bzw. Knoten-IDs in verschiedenen Partitionen (im Folgenden *gemeinsame Links* genannt) führen. Die zuvor vorgestellten Metriken, die Cut-Size und $(\lambda - 1)$ Metrik, tragen zu diesem Ziel bei. Beide minimieren die Anzahl der Hyperkanten (respektive Knoten) und damit indirekt die Anzahl der Links. Theoretisch kann eine Hyperkante zwar beliebig viele Knoten und damit Links enthalten, in der Praxis sind aber nur wenige Hyperkanten so groß. Die gleiche Argumentation gilt in analoger Weise für Knoten im Falle eines Vertex-Cuts.

Eine direktere Strategie, um gemeinsame Links zu vermeiden, müsste die Größe der Hyperkanten bzw. des Knotengrades berücksichtigen. Beispielsweise ließe sich die Summe der Kardinalitäten der Hyperkanten im Cut-Set bilden: $\sum_{e \in E} |e|$. Zum Zeitpunkt der Erstellung dieser Arbeit und nach bestem Wissen des Autors implementiert jedoch keines der verbreiteten Softwaretools eine solche Zielfunktion direkt. Eine Umsetzungsmöglichkeit besteht jedoch darin, die Kardinalität einer Hyperkante als Gewichtung dieser zu verwenden. Viele Verfahren unterstützen solche Wichtungen sowohl auf der zu partitionierenden als auch der zu balancierenden Menge. Die Zielfunktionen lassen sich dafür grundsätzlich leicht auf gewichtete (Hyper-) Graphen erweitern: die Cut-Size wird zur Summe über alle (Hyper-) Kanten- bzw. Knotengewichte. Das Communication Volume kann mit der Anzahl geschnittener Partitionen mal dem Gewicht der (Hyper-) Kante bzw. des Knotens gebildet werden.

Alternativ ließe sich auch auf den Darstellungen als Graph arbeiten, wobei sich für Vertex-, Edge-Cut und Hybrid-Cut unterschiedliche Ergebnisse zeigen. Für letzteren ist die bipartite Repräsenation grundsätzlich gut geeignet, da ein Vertex-Cut des bipartiten Graphen einem Hybrid-Cut des zugehörigen Hypergraphen entspricht. Hier lassen sich gemeinsame Links in der Zielfunktion ausdrücken, indem jedem Knoten des bipartiten Graphen die Anzahl der zu ihm inzidenten Kanten als Gewicht zugewiesen wird. Abb. 14 illustriert einen Hybrid-Cut eines Hypergraphen anhand dessen bipartiter Darstellung. Dabei ist die dargestellte Zerlegung nicht die einzig (sinnvoll) mögliche. Beispielsweise gibt es noch einen Cut in der Knotenmenge des Hypergraphen mit $(\{e1, e2\}, \{e3\}, \{e5, e6\})$ sowie einen durch Hyperkanten mit $(\{v1, v2, v5\}, \{v3, v4, v8\}, \{v7, v6\})$. Man beachte, dass alle Varianten in diesem Beispiel eine perfekte Balance für Knoten, Hyperkanten und Links gleicherma-

ßen liefern, wobei der Vertex-Cut die geringste Cut-Size erreicht.

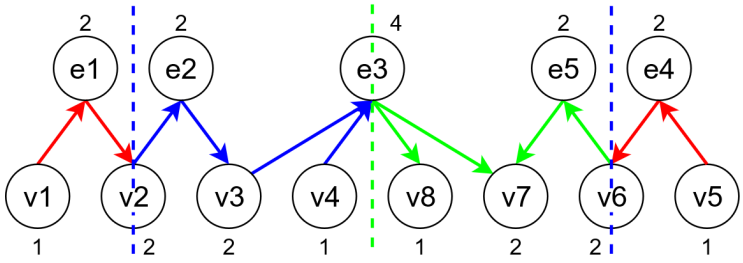


Abbildung 14: Der in Abb. 11 dargestellte Hybrid-Cut in bipartiter Repräsentation. An jedem Knoten ist sein Knotengrad annotiert. Kanten entsprechen Links und sind entsprechend ihrer Partition farbig markiert.

Heintz et. al. nutzen die bipartite Repräsentation und nehmen an, dass auch "eine Übertragung auf die Knotenmenge partitionierende Algorithmen in ähnlicher Weise umgesetzt werden könnte"[10] und sehen dies als potenziellen Gegenstand weiterer Arbeiten [Hei+16]. Man beachte jedoch, dass die Autoren das Mapping nur in eine Richtung, nämlich vom Hypergeaphen zum Graphen, durchführen. Für das WossiDiA-System ist jedoch ein bidirektionales von Nöten, da der Hypergraph selbst und nicht dessen Graphrepräsentation verteilt werden soll. Ein Edge-Cut der Star-Expansion bedeutet effektiv ein Durchtrennen der Links des Hypergraphen, was nicht zu einer Partitionierung der Links führt. Daher scheidet diese Variante für den vorliegenden Anwendungsfall aus.

In der Praxis kann ein Vertex- bzw. Edge-Cut des Hypergraphen dennoch, wiederum mittels Gewichten, aus einer Kantenpartitionierung der Star-Expansion gewonnen werden. Dazu belegt man die entsprechenden Knoten des bipartiten Graphen mit so großen Wichtungen, dass sie nicht geteilt werden. Die Zielfunktion wird also dazu genutzt, eine bestimmte Art der Partitionierung zu erzwingen. Dabei ist zu beachten, dass, wie zuvor festgestellt, durch die Festlegung auf Vertex- oder Edge-Cut die Qualität der gewonnenen Lösung im Vergleich zum Hybrid-Cut sinken kann.

Bei der Clique-Expansion zeichnet sich ein anderes Bild, da hier Hyperkanten nicht mehr explizit repräsentiert werden und sich damit eine Partitionierung der Hyperkanten nicht direkt ableiten lässt. Wie zuvor festgestellt kann eine Zerlegung der Knoten jedoch eins zu eins auf den Hypergraphen übertragen werden. Zwischen Cut-Size und der Anzahl gemeinsamer Links stellt sich dabei ein etwas

[10] Zitat frei aus dem Englischen übersetzt

komplexerer Zusammenhang als bei der bipartiten Variante heraus. Wie in Kapitel 4.2 festgehalten, skaliert die Anzahl der Kanten quadratisch mit der Größe einer Hyperkante. Damit ergeben sich für eine Teilung einer solchen $|e| = n+m$ genau $n+m$ gemeinsame Links, jedoch $n \cdot m$ geteilte Kanten in der Graphrepräsentation. Dabei bezeichnen n und m die Anzahl Knoten der Hyperkante auf beiden Seiten des Cuts. Dies bedeutet anschaulich, dass die Clique-Expansion das Teilen großer Hyperkanten überproportional stark bestraft. Hinzu kommt, dass die Darstellung bei solchen stark in ihrer Größe wachsen kann, was in der Praxis dazu führen kann, dass diese Variante nicht mit der zur Verfügung stehenden Hardware umsetzbar ist [Hei+16].

Ein weiterer, für den Anwendungsfall relevanter Aspekt ist der Einfluss von Richtung und Rolle der Links auf die Zielfunktion. Besonders letztere ist dabei von Interesse, da diese aussagekräftiger als die Richtung ist und zusätzliche semantische Informationen darüber enthält, wie Knoten an Hyperkanten teilnehmen. Diese Bedeutung lässt sich gut als Gewichte auf die Links übertragen, sodass bestehende Algorithmen bei der Partitionierung damit rechnen können. Denkbar wäre beispielsweise, dass Links, die einen örtlichen Zusammenhang modellieren, wichtiger sind, als solche, die eine zeitliche Beziehung darstellen. Allerdings arbeiten nicht alle Verfahren direkt auf den Links oder deren Repräsentation, wie z.B. die klassischen Verfahren für Hypergraphen, was ein Mapping der Gewichte auf Hyperkanten oder Knoten notwendig macht. Auch Knoten- und Kantentypen lassen sich auf diese Weise zur Partitionierung nutzen, wobei die Wichtung stehts auf die Elemente zu legen ist, die geteilt werden, d.h. Knoten beim Vertex-Cut und Hyperkanten im Falle eines Edge-Cut.

Wie eine konkrete Übertragung von Typen zu Gewichten aussieht, hängt stark davon ab, worauf der Fokus durchzuführender Anfragen liegt. Sie bieten jedoch ein gutes Werkzeug, um bei gleichbleibender Partitionierungsstrategie ein Feintuning vorzunehmen. Ein solches ist mit weniger Aufwand als eine vollständige Repartitionierung verbunden, da hier keine Relationen neu und anhand eines anderen Schemas zu erstellen sind. Ein solches Mapping von Typen auf Gewichte ist jedoch nicht nur von Nutzen, um Trennungen bestimmter Arten von Beziehungen zu vermeiden, sondern kann auch die Balancierung beeinflussen, was im Folgenden kurz diskutiert werden soll.

5.4 Art der Balancierung

Während die klassische Problemstellung der Graphpartitionierung die Balancierung direkt mit der zu partitionierenden Menge verknüpft, ist für den Anwendungsfall eine entkoppelte Betrachtung notwendig. Beispielsweise wäre bei einem Vertex-Cut eine ausgeglichene Verteilung von Knoten denkbar. Wie zuvor eignen sich auch hier einige Ansätze auf natürliche Weise. Klassische Verfahren sind auf die Balancierung von Knoten ausgelegt, neuere Ansätze können jedoch auch Hyperkanten balancieren. Die dritte Möglichkeit stellt eine Balancierung der Links dar, bei der sich eine Partitionierung der Kanten des zugehörigen bipartiten Graphen anbietet.

Insbesondere erstere Variante ist dabei im Hinblick auf die zuvor erarbeiteten Grenzen für den Worst-Case kritisch zu betrachten. Eine Gleichverteilung der Knoten kann zu sehr schlechten Verteilungen für Hyperkanten führen und damit Auswertungen der Topologie stark ausbremsen. Aber auch eine Balancierung von Hyperkanten kann potenziell eine schlechte Verteilung der Knoten bedeuten. Bei einer gleichmäßigen Verteilung der Links hingegen gelten die deutlich besseren Grenzen für Graphen, speziell sogar die für bipartite Graphen. Ohne weitere Annahmen über den Fokus der zu stellenden Anfragen (Daten- oder Topologiezentriert) zu treffen, lässt sich die Balancierung der Links zumindest aus theoretischer Sicht als die vorzuziehende Variante festhalten. Inwieweit diese theoretischen Grenzen sich im Fall des vorliegenden Hypergraphen praktisch auswirken, ist jedoch offen und muss anhand konkreter Umsetzungen bewertet werden.

Analog zur Anwendung bei der zu optimierenden Metrik lässt sich auch die Typisierung durch eine Gewichtung in die Balancierung einbringen. Sollen beispielsweise Knoten balanciert werden, sodass insbesondere Orte gleichmäßig verteilt sind, können diese Knoten mit einem höheren Gewicht versehen werden. Dieses Prinzip lässt sich analog für Hyperkanten oder Links anwenden. Damit ist auch bezüglich der Balancierung ein Mittel zum Finetuning der Verteilung gegeben.

5.5 Zusammenfassung und umsetzbare Strategien

Die differenzierte Betrachtung der einzelnen Schwerpunkte des Problems führt zu einer Vielzahl möglicher Strategien mit jeweils individuellen Vor- und Nachteilen sowie verschiedenen Umsetzungsmöglichkeiten. Im Kern lassen sich die

Optionen wie folgt zusammenfassen:

Cut:	Vertex-, Edge- oder Hybrid-Cut
Cut-Metrik:	Cut-Size oder $(\lambda - 1)$ Metrik, ggf. mit Gewichten
Balancierung:	Knoten, Hyperkanten oder Links, ggf. mit Gewichten
Allokation:	Replikationen nutzen oder nicht nutzen

Nicht alle Kombinationen lassen sich direkt mit bestehenden Verfahren umsetzten. Die klassische Variante *"Edge-Cut – Communication Volume – Knotenbalanciert – mit Replikationen"* lässt sich relativ leicht mit vorhandenen Verfahren umsetzten, wobei mit Replikationen solche für die Datentabellen gemeint sind. Für einen Vertex-Cut gibt es jedoch kaum Algorithmen für Hypergraphen. Hier, und besonders bei einem Hybrid-Cut, wird ein bidirektionales Mapping vor allem zur Star-Expansion relevant. Dasselbe trifft für die Balancierung von Hyperkanten oder Links zu. Leicht umzusetzten sind außerdem Varianten, in denen das nicht durchtrennte Element balanciert wird, d.h. Vertex-Cut und Hyperkanten balanciert, Edge-Cut und Knoten balanicert oder aber Hybrid-Cut und Links balanciert.
Darüber hinaus sind nicht alle Kombinationen gleichermaßen sinnvoll. So ist das Communication Volume als Metrik nicht geeignet, wenn Replikationen genutzt werden. Solche profitieren nämlich von Knoten (Hyperkanten), die in vielen Partitionen vorkommen, da dies zu weniger unnötig replizierten Datensätzen führt. Dies wird jedoch durch die $(\lambda - 1)$ Metrik bestraft.

Zusammenfassend lässt sich feststellen, dass verschiedene Strategien zur Verteilung des Hypergraphen denkbar sind und sich jeweils einige für spezielle Zielstellungen besser eignen als andere. Das klassische Vorgehen ist besonders für knotenzentrierte Anfragen sowie Forward- und Backward-Star Berechnungen geeignet, während sich ein Vertex-Cut mit balancierten Hyperkanten besonders für hyperkantenzentrierte Abfragen und Nachbarschaftsberechnungen anbietet. Einen Mittelweg bietet ein Hybrid-Cut mit balancierten Links, der sich besonders guter Eigenschaften bezüglich der letztendlichen Lastbalancierung erfreut.

6 UMSETZUNG UND FAZIT

Von den im vorigen Kapitel beschriebenen Varianten sollen nun zwei Ansätze auch anhand einer praktischen Umsetzung bewertet werden. Dazu wurde der in WossiDiA gespeicherte Hypergraph mit dem frei verfügbaren hMETIS [KK95] durch einen Edge- sowie Vertex-Cut partitioniert. Das Softwaretool stellt Multilevel-Verfahren bereit, um Hypergraphen in ihrer Knotenmenge zu partitionieren, wobei die Cut-Size oder auch die *Sum of external Degrees (SoeD)* minimiert wird. Letztere ist definiert als Summe über alle einzelnen Cut-Sizes und unterscheidet sich damit von der $(\lambda - 1)$ Metrik nur darin, dass eine k Partitionen schneidende Hyperkante mit der Wertigkeit k anstatt mit $k-1$ zur Gesamtsumme beiträgt.

6.1 Bestimmung eines Vertex-Cuts

Um beide Cut-Varianten mit dem Tool umzusetzen bedarf es jedoch noch einer weiteren Überlegung, denn nur eine Partitionierung der Knoten ist direkt möglich. Um aber auch eine Zerlegung der Hyperkanten zu ermöglichen, transformieren wir den Hypergraphen so, dass jede Hyperkante durch einen Knoten und jeder Knoten durch eine Hyperkante repräsentiert wird und vertauschen so effektiv die beiden Mengen. Dass dies ohne Probleme möglich und tatsächlich eine bidirektionale Zuweisung ist, lässt sich leicht in der Darstellung der Inzidenzmatrix oder der des entsprechenden bipartiten Graphen feststellen. Bei letzterem bedeutet dies, dass die beiden Teile des bipartiten Graphen ihre Bedeutung tauschen. Man beachte, dass dies nur möglich ist, da der Hypergraph zusammenhängend ist und damit keine isolierten Knoten enthält. Solche würden nämlich in nicht zulässigen, leeren Hyperkanten resultieren. Mit dieser Umwandlung bleiben auch die gewünschten zu optimierenden Eigenschaften erhalten. Der Knotengrad wird zur Kardinalität der neuen Hyperkante und vice versa, weiterhin entsteht für jede Hyperkante genau ein Knoten und umgekehrt, sodass dies auch auf die Balancierung keine Auswirkungen hat. Darüber hinaus ist dieses Vorgehen aus praktischer Sicht sehr komfortabel, da eine "Umbenennung" der beiden Mengen bereits ausreichend ist. Abb. 15 beleuchtet diese Eigenschaften an der bipartiten Darstellung eines Beispielhypergraphen. Mit dieser Argumentation ist jedoch nur gesichert, dass die Algorithmen korrekt arbeiten können und ein aussagekräftiges Ergebnis für den wahren Hypergraphen abgeleitet werden kann. Ob jedoch die genutzten Heuristiken, die auf typische Hypergraphen zugeschnitten sind, auch hier zu einer qualitativ guten Lösung führen, ist anhand konkreter Ergebnisse zu beurteilen. Diese sollen im Folgenden für den

WossiDiA-Hypergraphen erörtert werden.

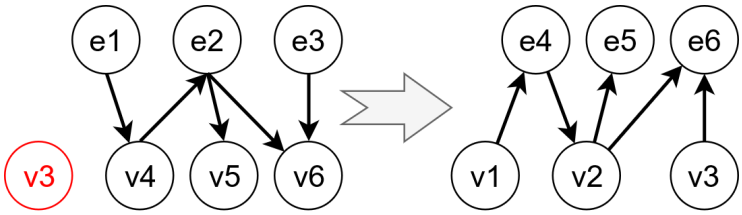


Abbildung 15: Bipartite Repräsentation eines kleinen Hypergraphen. Werden $v4$ bis $v6$ als Hyperkanten interpretiert, ist deren Knotengrad als Kardinalität anzusehen. Umgekehrtes gilt für die Hyperkanten $e1$ bis $e3$. Damit wird beispielsweise aus $v4$, einem Knoten mit $deg(v4) = 2$, eine Hyperkante mit $|v4| = 2$. Für $v3$ kann dies nicht umgesetzt werden, da der Knoten zu einer leeren Hyperkante würde.

6.2 Ergebnisse der Umsetzung

Technische Details

Für die Berechnungen wurde das Programm *khmetis* in der Version 1.5.3 verwendet. Als zugrundeliegendes System stand ein Windows 10 (Version 1709) Laptop mit einem Core i5-6200U @ 2,3 GHz (Boost 2,4 GHz) und 8 GB RAM zur Verfügung. Die Hypergraphstruktur wurde aus der Link-Tabelle des WossiDiA-Systems in zwei CSV-Dateien exportiert, aufgeschlüsselt nach Knoten bzw. Hyperkanten. Für beide Cut-Varianten wurden 12 Zerlegungen in $k = 3$ bis $k = 15$ Partitionen bestimmt, wobei für jede Anzahl beide Optimierungsfunktionen zum Einsatz kamen. Der Balancierungsfaktor wurde mit 1,1 (10 Prozent) vorgegeben und für jede Kombination 10 Wiederholungen angesetzt, aus denen die beste gewählt wurde.

Die Ergebnisse sind in Tabelle 3 für 5 ausgewählte Werte von k dargestellt.

Art der Partitionierung

Die Ergebnisse zeigen klar, dass ein Vertex-Cut dem klassischen Edge-Cut bezüglich der gemessenen Größen deutlich überlegen ist. Über alle Metriken und Partitionsgrößen hinweg führt eine Zerlegung der Knoten zu Resultaten, die etwa um den Faktor 5 schlechter sind, als die der Hyperkantenzerlegung. Selbst unter Berücksichtigung der relativen Anteile des Cut-Sets zur Basismenge bleibt der Unterschied groß. Dieses Ergebnis ist konsistent mit früheren Erkenntnis-

		$k=5$	$k=8$	$k=10$	$k=12$	$k=14$
Vertex-Cut	Cut-Size *(opt)*	5.689	6.254	6.355	6.480	6.543
	SoeD	16.568	22.078	24.139	26.048	27.680
	Cut-Size	6.002	6.865	7.208	7.440	7.776
	SoeD *(opt)*	16.101	20.748	22.778	24.563	26.169
Edge-Cut	Cut-Size *(opt)*	33.067	42.039	46.243	49.775	52.441
	SoeD	68.859	87.799	97.412	105.163	111.015
	Cut-Size	33.647	42.163	46.714	50.161	53.002
	SoeD *(opt)*	69.439	87.510	97.450	104.691	110.647

Tabelle 3: Die Ergebnisse mehrerer Zerlegungen mittels *khmetis*. Welche Metrik als Zielfunktion diente, ist in Klammern gesetzt. Der Edge-Cut ist durchgehend deutlich schlechter für beide Metriken und Optimierungsziele. Partitioniert wurden 257.462 Knoten und 339.819 Hyperkanten.

sen, die für (Hyper-) Graphen mit verzerrtem, reziprok exponentialverteilten Knotengrad eine Partitionierung der Kanten dem klassischen Ansatz vorziehen. Tatsächlich folgt auch der WossiDiA-Hypergraph einem solchen Schema, wie Abb. 16 zeigt.

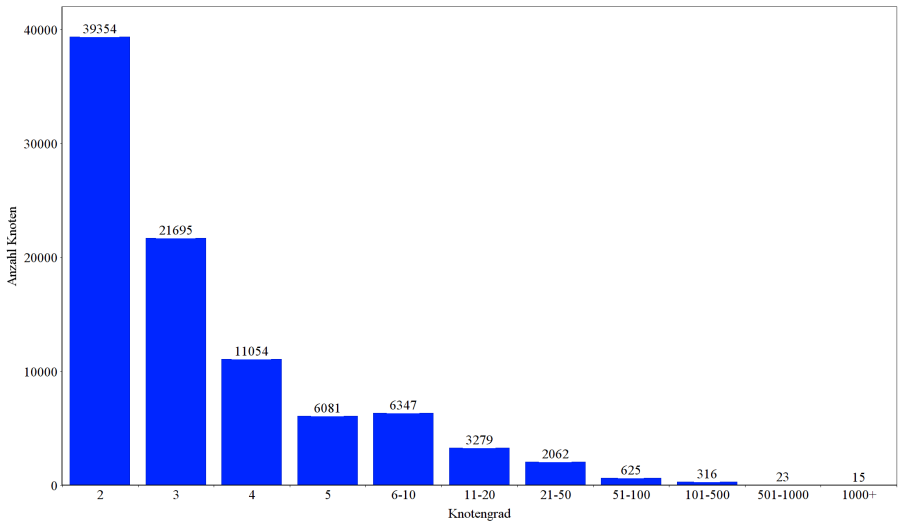


Abbildung 16: Verteilung der Knoten im WossiDiA-Hypergraphen, aufgetragen über dem Knotengrad. Die Grafik lässt den reziproken exponentiellen Verlauf der Verteilung erkennen. Knoten mit Grad 1 sind nicht dargestellt, stellen aber die mit Abstand größte Teilmenge mit 162.137 Elementen dar.

Optimierungsziele

Kaum Differenzen zeigen sich hingegen beim Vergleich der beiden Optimierungsziele. Bei beiden Cut-Varianten lassen sich leichte Verschiebungen zur jeweils optimierten Metrik hin feststellen, große Sprünge lassen sich jedoch nicht erkennen. Gleiches gilt für die Abhängigkeit von der Anzahl der Partitionen. Für den untersuchten Bereich zwischen 3 und 15 Partitionen stellt sich keine Anzahl als deutlich besser im Vergleich zu anderen heraus. Stattdessen wachsen die erfassten Metriken moderat linear mit zunehmendem k. Diese Zunahme des Cut-Sets bzw. der Sum of external Degrees zeigt lediglich den natürlichen Kompromiss zwischen dem Grad der Zerlegung und dem Anteil durchtrennter Strukturen, der für den Anwendungsfall gleichbedeutend mit dem Abwägen zwischen Parallelität und Kommunikationsaufwand ist.

Balance

Neben den Ergebnissen bezüglich der Cut-Metriken ist natürlich auch die resultierende Balance relevant. Diese stellt sich bei allen Varianten durchgehend als hervorragend heraus. Keine der Zerlegungen schöpft die möglichen 10 Prozent voll aus, das Maximum liegt bei ca. 6%. Von den Vertex-Cuts erreichen 18 der 24 bestimmten Partitionierungen sogar einen Wert $< 0.5\%$, im Falle des Edge-Cuts sind es immerhin 10. Damit ist aus Sicht der Lastverteilung bezüglich der partitionierten Menge kein Verfahren besonders vorzuziehen.

Laufzeit

Das genutzte Multilevel-Verfahren zeichnet sich darüber hinaus auch durch kurze Laufzeiten aus, Abb. 17 stellt diese grafisch dar. Auch in diesem Punkt schneidet der Vertex-Cut deutlich besser ab als der Edge-Cut. Beide Ansätze zeigen wiederum ein moderates, lineares Wachstum mit der Anzahl an Partitionen. Dabei ist zu bedenken, dass die angegebenen Zeiten jeweils 10 Wiederholungen beinhalten und auf relativ schwacher Hardware gemessen wurden. Weniger Iterationen und leistungsfähigere Hardware würden die Ausführungszeiten wahrscheinlich reduzieren. Damit schließen diese Ergebnisse auch eine "online" vorgenommene Repartitionierung im WossiDiA-System nicht aus, bei der größere Änderungsoperationen am Hypergraphen unmittelbar mit einer Neuverteilung verknüpft werden.

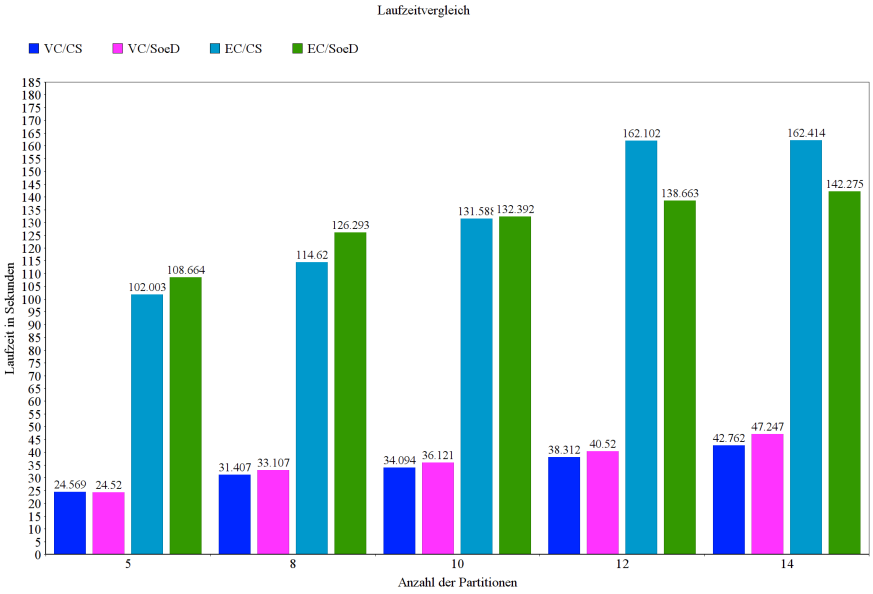


Abbildung 17: Laufzeiten der Berechnungen für ausgewählte Werte von k. *VC*, *EC* und *CS* stehen für Vertex-, Edge-Cut und Cut-Size.

	Vertex-Cut		*Edge-Cut*	
	Cut-Size	*SoeD*	*Cut-Size*	*SoeD*
Durchschnitt	91.163	306.569	307.766	642.382
Minimum	91.119	306.357	307.704	642.360
Maximum	91.207	306.781	307.828	642.404

Tabelle 4: Ergebnisse einer zufälligen Verteilung über die Knoten- bzw. Kanten-IDs auf 10 Partitionen.

Vergleich mit zufälliger Verteilung

Im Vergleich heben sich beide Varianten deutlich von einer zufällig vorgenommenen Verteilung ab. Eine solche wurde innerhalb der PostgreSQL-Datenbank, die den Hypergraphen speichert, mithilfe der *random()* Funktion vorgenommen. Bei einer solchen, pseudo-zufälligen Zerlegung der Knoten- bzw. Kantenmenge in $k = 10$ Partitionen und 10 Wiederholungen beläuft sich das durchschnittliche Cut-Set auf 91.163 Knoten bzw. 307.766 Hyperkanten. Einen Überblick über die Ergebnisse einer solchen Verteilung bietet Tabelle 4. Eine zufällige Partitionierung der Knoten führt offenbar zu derartig schlechten Zerlegungen, dass fast alle ($> 90\%$) Hyperkanten im Cut-Set liegen, bei einem Vertex-Cut sind es immerhin noch ca. ein Drittel. Damit wird die Notwendigkeit geeigneter Algorithmen abermals deutlich.

Übertragung in eine Datenbank

Um eine Partitionierung in der Datenbank umzusetzen sind neben der eigentlichen Berechnung einer Zerlegung weitere Schritte notwendig. Eine ggf. beim Export der Struktur vorgenommene Neuzuweisung von IDs ist beim Import der Ergebnisse zu berücksichtigen. Letztere liegen typischerweise in Form einer Liste von Knoten bzw. Kanten und der Nummer ihrer zugehörigen Partition vor. Daraus ist wiederum eine Neuverteilung der IDs zu berechnen, um die Partitionierung in der Datenbank mit Bereichen umsetzten zu können. Das gilt für alle zu partitionierenden Relationen. Entsprechend der Anzahl Partitionen sind dann alle Tabellen mit den korrekten Bereichen neu zu erstellen und aufzufüllen. Für das WossiDiA-System und eine Partitionierung mit khmetis bedeutet das folgende konkrete Schritte:

1. Neuzuweisung der Knoten- und Kanten-IDs in die Intervalle $[1, |V|]$ und $[1, |E|]$
2. Export in entsprechendes Format und Berechnung einer k-Wege-Zerlegung
3. Ableitung einer Zerlegung für die nicht partitionierte Menge
4. Neuzuweisung der Knoten- und Kanten-IDs in k disjunkte Intervalle entsprechend ihrer Partitionen
5. Neuerstellung der Link- sowie aller Datentabellen mit einer, den zuvor bestimmten Intervallen entsprechenden Partitionierung
6. Import der Daten in die neu erstellten Relationen

Diese Schritte, wenn manuell vorgenommen, sind mit einem nicht unerheblichem Aufwand verbunden, daher bieten sich hier gespeicherte Datenbank-Prozeduren an, insbesondere, wenn mehrfach repartitioniert werden soll. Zukünftige Arbeiten könnten darauf aufbauend die Auswirkungen verschiedener Umsetzungen durch praktische Untersuchungen innerhalb einer Datenbank nachvollziehen.

6.3 Zusammenfassung

Das Problem, eine für die Parallelisierung von Anfragen geeignete Partitionierungsstrategie für Hypergraphen zu bestimmen und umzusetzen, ist durchaus komplex und facettenreich. Neben der herkömmlichen Herangehensweise einer balancierten Partitionierung der Knoten wurden in dieser Arbeit auch eine Zerlegung der Hyperkanten sowie eine hybride Lösung vorgestellt. Darauf aufbauend

wurden für den Anwendungsfall Optimierungsmetriken, Gewichtungen, Strategien für die Allokation sowie Verfahren zur Umsetzung diskutiert. Klassische Techniken im Bereich der Hypergraphpartitionierung stellen sich bei einer Übertragung auf diese Problemstellung als nachteilig gegenüber anderen, in diesem Kontext noch wenig erforschten Ansätzen heraus. Dies zeigt sich sowohl auf konzeptioneller als auch praktischer Ebene. Die Anwendung in einem verteilten Datenbanksystem bringt dabei ihre eigenen Anforderungen mit sich, die sich von denen herkömmlicher Anwendungsfälle unterscheiden, beispielsweise in den Punkten Balancierung, Replikation oder Kostenfunktion. Diese haben Auswirkungen auf für das Partitionierungsproblem zu nutzenden Optimierungs- und Balancierungsanforderungen. So ist für die Parallelisierung von Anfragen sowohl die Balancierung von Hyperkanten als auch Knoten von Bedeutung. Und während bei anderen Anwendungsfällen die Vermeidung von Knotenreplikationen durch Minimierung des Communication Volume eine wichtige Rolle spielt, stellt sich diese im Kontext einer verteilten Datenbank nicht als zielführend heraus.

Die Ergebnisse zeigen aber auch weitere Forschungsmöglichkeiten auf. Dazu gehört beispielsweise ein Vergleich verschiedener praktischer Umsetzungen in einer verteilten Datenbank. Aber auch eine praktische Gegenüberstellung der Ergebnisse mit denen eines Hybrid-Cuts ist denkbar. Diese Arbeit bietet zumindest Anhaltspunkte, um weitere Strategien umzusetzen und zu vergleichen.

LITERATUR

[ABT12] James Abello, Peter Broadwell und Timothy R. Tangherlini. "Computational Folkloristics". In: *Commun. ACM* 55.7 (Juli 2012), S. 60–70. ISSN: 0001-0782. DOI: 10.1145/2209249.2209267.

[AIV15] Dan Alistarh, Jennifer Iglesias und Milan Vojnovic. "Streaming Min-max Hypergraph Partitioning". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'15. Montreal, Canada: MIT Press, 2015, S. 1900–1908.

[AL17] Giorgio Ausiello und Luigi Laura. "Directed hypergraphs: Introduction and fundamental algorithms—A survey". In: *Theoretical Computer Science* 658 (2017). Horn formulas, directed hypergraphs, lattices and closure systems: related formalism and application, S. 293 –306. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/j.tcs.2016.03.016.

[AR04] Konstantin Andreev und Harald Räcke. "Balanced Graph Partitioning". In: *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '04. Barcelona, Spain: ACM, 2004, S. 120–124. ISBN: 1-58113-840-7. DOI: 10.1145/1007912.1007931.

[ARK06] Amine Abou-Rjeili und George Karypis. "Multilevel Algorithms for Partitioning Power-law Graphs". In: *Proceedings of the 20th International Conference on Parallel and Distributed Processing*. IPDPS'06. Rhodes Island, Greece: IEEE Computer Society, 2006, S. 124–124. ISBN: 1-4244-0054-6.

[Ber89] Claude Berge. *Hypergraphs — Combinatorics of Finite Sets*. 1st Edition. North Holland, 1989.

[BLV14] Florian Bourse, Marc Lelarge und Milan Vojnovic. "Balanced Graph Edge Partition". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: ACM, 2014, S. 1456–1465. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623660.

[Bre13] Alain Bretto. *Hypergraph Theory*. Springer, Heidelberg, 17. Apr. 2013. 119 S. ISBN: 978-3-319-00080-0.

[Bul+13] Aydin Buluç u. a. "Recent Advances in Graph Partitioning". In: *CoRR* abs/1311.3144 (2013).

[CA01] U. V. Catalyurek und C. Aykanat. “A Hypergraph-Partitioning Approach for Coarse-Grain Decomposition”. In: *Supercomputing, ACM/IEEE 2001 Conference*. 2001, S. 42–42. DOI: 10.1109/SC.2001.10035.

[Cat+07] U. V. Catalyurek u. a. “Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations”. In: *2007 IEEE International Parallel and Distributed Processing Symposium*. 2007, S. 1–11. DOI: 10.1109/IPDPS.2007.370258.

[Che+14] Rong Chen u. a. “Bipartite-oriented Distributed Graph Partitioning for Big Learning”. In: *Proceedings of 5th Asia-Pacific Workshop on Systems*. APSys ’14. Beijing, China: ACM, 2014, 14:1–14:7. ISBN: 978-1-4503-3024-4. DOI: 10.1145/2637166.2637236.

[Che+15] Rong Chen u. a. “PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs”. In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys ’15. Bordeaux, France: ACM, 2015, 1:1–1:15. ISBN: 978-1-4503-3238-5. DOI: 10.1145/2741948.2741970.

[Dev+02] Karen Devine u. a. “Zoltan Data Management Services for Parallel Dynamic Applications”. In: *Computing in Science and Engineering* 4.2 (2002), S. 90–97.

[Dev+06] K. D. Devine u. a. “Parallel hypergraph partitioning for scientific computing”. In: *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*. 2006, 10 pp.–. DOI: 10.1109/IPDPS.2006.1639359.

[Fie73] Miroslav Fiedler. “Algebraic connectivity of graphs”. In: *Czechoslovak mathematical journal* 23.2 (1973), S. 298–305.

[FM88] Charles M Fiduccia und Robert M Mattheyses. “A linear-time heuristic for improving network partitions”. In: *Papers on Twenty-five years of electronic design automation*. ACM. 1988, S. 241–247.

[GLP93] Giorgio Gallo, Giustino Longo und Stefano Pallottino. “Directed Hypergraphs and Applications”. In: *Discrete Applied Mathematics* 42.2 (1993), S. 177–201. DOI: 10.1016/0166-218X(93)90045-P.

[Gon+12] Joseph E. Gonzalez u. a. “PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs”. In: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX, 2012, S. 17–30. ISBN: 978-1-931971-96-6.

[Hei+16] Benjamin Heintz u. a. “MESH: A Flexible Distributed Hypergraph Processing System”. In: (2016).

[HZY15] Jin Huang, Rui Zhang und Jeffrey Xu Yu. “Scalable Hypergraph Learning and Processing”. In: *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*. ICDM ’15. Washington, DC, USA: IEEE Computer Society, 2015, S. 775–780. ISBN: 978-1-4673-9504-5. DOI: 10.1109/ICDM.2015.33.

[Kab+17] Igor Kabiljo u. a. “Social Hash Partitioner: A Scalable Distributed Hypergraph Partitioner”. In: *CoRR* abs/1707.06665 (2017).

[KK95] George Karypis und Vipin Kumar. *METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*. Techn. Ber. 1995.

[KK99] George Karypis und Vipin Kumar. “Multilevel K-way Hypergraph Partitioning”. In: *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*. DAC ’99. New Orleans, Louisiana, USA: ACM, 1999, S. 343–348. ISBN: 1-58113-109-7. DOI: 10.1145/309847.309954.

[KL70] B. W. Kernighan und S. Lin. “An efficient heuristic procedure for partitioning graphs”. In: *The Bell System Technical Journal* 49.2 (1970), S. 291–307. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1970.tb01770.x.

[Lan04] Kevin Lang. “Finding good nearly balanced cuts in power law graphs”. In: (Dez. 2004).

[Les+08] Jure Leskovec u. a. “Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters”. In: *CoRR* abs/0810.1355 (2008). arXiv: 0810.1355.

[Li+17] Lingda Li u. a. “A Simple Yet Effective Balanced Edge Partition Model for Parallel Computing”. In: *Proc. ACM Meas. Anal. Comput. Syst.* 1.1 (Juni 2017), 14:1–14:21. ISSN: 2476-1249. DOI: 10.1145/3084451.

[MSH17] Holger Meyer, Alf-Christian Schering und Andreas Heuer. “The Hydra.Powergraph System — Building digital archives with directed and typed hypergraphs”. In: *Datenbank-Spektrum* 17.3 (2017). DOI: 10.1007/s13222-017-0253-x. Eingereicht.

[MSS14] Holger Meyer, Alf-Christian Schering und Christoph Schmitt. “WossiDiA — The Wossidlo Digital Archive”. In: *Corpora Ethnographica Online*. 2014.

[PM07] David A Papa und Igor L Markov. “Hypergraph Partitioning and Clustering.” In: (2007).

[Pre14] Alon Presta Alessandro Shalita. “Large-scale graph partitioning with Apache Giraph”. In: (Apr. 2014).

[Sto+10] Michael Stonebraker u. a. “MapReduce and Parallel DBMSs: Friends or Foes?” In: *Commun. ACM* 53.1 (Jan. 2010), S. 64–71. ISSN: 0001-0782. DOI: 10.1145/1629175.1629197.

[STT07] Ekaterina Smorodkina, Mayur Thakur und Daniel Tauritz. “Algorithms for the Balanced Edge Partitioning Problem”. In: *Experimental Algorithms: 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007. Proceedings*. Hrsg. von Camil Demetrescu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 311–323. ISBN: 978-3-540-72845-0. DOI: 10.1007/978-3-540-72845-0_24.

[TK08] Aleksandar Trifunović und William J. Knottenbelt. “Parallel multilevel algorithms for hypergraph partitioning”. In: *Journal of Parallel and Distributed Computing* 68.5 (2008), S. 563 –581. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2007.11.002.

[WW93] Dorothea Wagner und Frank Wagner. “Between Min Cut and Graph Bisection”. In: *Mathematical Foundations of Computer Science 1993: 18th International Symposium, MFCS'93 Gdańsk, Poland, August 30–September 3, 1993 Proceedings*. Hrsg. von Andrzej M. Borzyszkowski und Stefan Sokołowski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, S. 744–750. ISBN: 978-3-540-47927-7. DOI: 10.1007/3-540-57182-5_65.

[Xin+14] Reynold S. Xin u. a. “GraphX: Unifying Data-Parallel and Graph-Parallel Analytics”. In: *CoRR* abs/1402.2394 (2014).

[Yan+16] Wenyin Yang u. a. “A Distributed Algorithm for Balanced Hypergraph Partitioning”. In: *Advances in Services Computing: 10th Asia-Pacific Services Computing Conference, APSCC 2016, Zhangjiajie, China, November 16-18, 2016, Proceedings*. Hrsg. von Guojun Wang, Yanbo Han und Gregorio Martínez Pérez. Cham: Springer International Publishing, 2016, S. 477–490. ISBN: 978-3-319-49178-3. DOI: 10.1007/978-3-319-49178-3_36.

[Zha+17a] Chenzi Zhang u. a. “Graph Edge Partitioning via Neighborhood Heuristic”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, S. 605–614.

[Zha+17b] Zhihong Zhang u. a. "Joint hypergraph learning and sparse regression for feature selection". In: *Pattern Recognition* 63.Supplement C (2017), S. 291 –309. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2016.06.009.

[ŠS06] Jiří Šíma und Satu Elisa Schaeffer. "On the NP-Completeness of Some Graph Cluster Measures". In: *SOFSEM 2006: Theory and Practice of Computer Science: 32nd Conference on Current Trends in Theory and Practice of Computer Science, Merin, Czech Republic, January 21-27, 2006. Proceedings*. Hrsg. von Jiří Wiedermann u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 530–537. ISBN: 978-3-540-32217-7. DOI: 10.1007/11611257_51.

DANKSAGUNG

Herrn Prof. Dr. A. Heuer danke ich für die Überlassung des Themas, sowie das stete Interesse am Fortgang der Arbeit.

Herrn Dr.-Ing. H. Meyer danke ich für die umfangreiche Betreuung der Arbeit, sowie die zahlreichen hilfreichen Diskussionen.

Darüber hinaus danke ich den Mitarbeiterinnen und Mitarbeitern des Lehrstuhls Datenbanken und Informationssysteme der Informatik, die mit Anregungen und Hilfestellungen zu dieser Arbeit beigetragen haben.

Weiterhin gilt mein herzlicher Dank allen, die Zwischenstände der Arbeit gelesen und mir wichtige Hinweise gegeben haben. Hier unterstützten mich Dr.-Ing. H. Meyer, Dipl. Ing. A. Schubert, Prof. Dr. J.-Ch. Kuhr und F. Weckmann.

Nicht zuletzt möchte ich mich bei meiner Freundin und meiner Familie für ihre Anteilnahme, Verständnis und Geduld bedanken.