Clustering von Bahnweichen und Analyse von Störungen zur Optimierung der Instandhaltungsmaßnahmen

Diplomarbeit

zur Erlangung des Grades

Diplom-Wirtschaftsmathematiker Univ.
im Studiengang Wirtschaftsmathematik

am Department Mathematik der
Friedrich-Alexander-Universität Erlangen-Nürnberg

vorgelegt am 16.03.2015

von **Thomas Hopf**

Betreuer: Prof. Dr. Frauke Liers

Betreuer: Dipl.-Ing.-Inf. Thomas Böhm

Inhaltsverzeichnis

Zusammenfassung

Nicht selten sind es Störungen von Eisenbahnweichen, die zu Verzögerungen innerhalb des Bahnverkehrs führen. In dieser Arbeit sollen die Daten eines Diagnosesystems für Weichenanlagen untersucht und ausgewertet werden, um für die Instandhaltung wertvolle Erkenntnisse zu liefern. Dabei finden zunächst Clusteranalyseverfahren Anwendung, welche die Menge der untersuchten Weichen in homogone Gruppen segmentiert. Dem schließt sich die Störungsanalyse an, wobei im letzten Schritt geprüft wird ob die Formation der Störungsdaten kongruent auf die zuvor gebildeten Objektstrukturen abgebildet werden kann.

Abkürzungsverzeichnis

AM Asset Management

API Application Programming Interface

DB AG Deutsche Bahn AG

DLR Deutsche Zentrum für Luft- und Raumfahrt

KDD Knowledge Discovery in Databases

LPA Label Propagation Algorithmus

PAM Plant Asset Management

VNS Variable Neighborhood Search

Kapitel 1

Einleitung

Weichen gehören im Netz der Deutsche Bahn AG (DB AG) zur Kategorie der beweglichen Fahrwegelemente und erlauben den Zügen im Schienenverkehr die Änderung der Fahrtrichtung ohne dabei die Fahrt unterbrechen zu müssen [6]. Innerhalb der Eisenbahninfrastruktur sind sie daher für den Verkehrsfluss von immenser Bedeutung.
Um eine hohe Verfügbarkeit des Schienennetzes zu gewährleisten ist eine funktionierende und effiziente Instandhaltung vonnöten. Statistiken belegen, dass Ausfälle und Störungen von Weichen einen äußerst kritischen Faktor im Eisenbahnbetrieb der DB AG darstellen. So ist im Jahr 2010 ein Anteil 19% der Zeitspanne sämtlicher Zugverspätungen auf Weichenstörungen zurückzuführen [23]. Dies entspricht kumuliert einer Dauer von 53,32 Jahren [9].

In dieser Arbeit soll eine Analyse der Funktionalität von Eisenbahnweichen, ausgestattet mit einem Diagnosesystem durchgeführt werden. Zunächst werden die Umstände und Potenziale der Weichendiagnostik unter Berücksichtigung des eingesetzten Systems erläutert, um anschließend die gesammelten Daten einordnen und untersuchen zu können. Im nächsten Schritt wird sich der Gruppe von Objekten gewidmet, welche existierende Weichen repräsentieren. Bei der Untersuchung soll ihr Verhalten auf die Ausprägungen der Attribute, unterteilt in Mess- und Weichenstammdaten zurückgeführt werden, weshalb auch näher auf die Charakteristika der Weichenanlagen eingegangen wird. Anschließend werden die bisherigen Erkenntnisse über Faktoren und Rollen in statistische Größen überführt, um die notwendigen Operationen und Funktionen der folgenden Datenanalyse zu ermöglichen.
Es wird mit einer Partitionierung der Objektmenge fortgefahren. Dabei kommen Verfahren der Clusteranalyse zum Einsatz, die Objekte in eine feste Zahl möglichst homogener Cluster zusammenfasst. Schließlich erhalten die Daten der auftretenden Weichenstörungen Einzug und es wird geprüft inwiefern die Ausfälle passend auf die Segmentierungen der Clusterlösungen abgebildet werden können.

Mit dem Datenanalysetool KNIME übernimmt eine quelloffene, auf der Entwicklungsumgebung Eclipse aufbauende Softwarelösung die praktische Durchführung der Analyse. Das jeweils letzte Unterkapitel der relevanten Analysedisziplinen erklärt die Implementierung der zuvor in der Theorie dargelegten Ansätze.
KNIME ist ein speziell auf Data-Mining ausgelegtes Tool, welches dank vorhandener Application Programming Interface (API) unkompliziert um neue Funktionen, Operatoren und Features erweitert werden kann. Es verfolgt das *Konzept des Modularen Pipelining*, indem Knoten (*Module*), welche mit den Daten operieren unter Zuhilfenahme von Kanten verknüpft werden. Die Schritte des Analyseprozesses werden auf eine Route, bestehend aus verbundenen Modulen abgebildet. Dabei können die Daten wahlweise aufbereitet, transformiert, gefiltert, gruppiert bzw. aggregiert, statistisch analysiert oder graphisch präsentiert werden.
Die Routen als Netzwerke von Knoten und Kanten werden in einem in sich geschlossenen Workflow konfiguriert. Jeder Workflow hat somit eine Initialisierungsphase in Form eines oder mehrerer sogenannter „Reader“-Knoten, die das Einlesen der Datenquelle übernehmen. Weiterhin erfolgt der Abschluss einer Route mit Hilfe von „Writer“-Knoten, die das (Zwischen-)Ergebnis des Workflows sichern und als Input für fortführende Workflows verfügbar machen. Im Sinne des Data-Mining verfolgen die Workflows im Verbund das Ziel Muster in den Daten zu erkennen und Informationen bzw. Wissen zu entdecken, so dass betriebswirtschaftlich relevante Thesen aufgestellt werden können.

Kapitel 2

Weichendiagnostik

Weichen, als Außenanlagen in der Eisenbahninfrastruktur zählt man ebenso wie Gleissperren zur Gruppe der *beweglichen Fahrwegelemente* [15]. Sie ermöglichen die Verbindung von mindestens zwei Gleisen im Bahnverkehr, wobei der Fahrweg durch die Zungenstellung der Weiche vorgegeben wird. Die korrekte Zungenendlage von Bahnweichen wird vor der Überfahrt im Zuge der Schutzfunktion *Sicherung beweglicher Fahrwegelemente* sichergestellt und die Eisenbahn-Bau- und Betriebsordnung [7] präzisiert: „Weichen, die gegen die Spitze befahren werden, müssen von den für die Zugfahrt gültigen Signalen derart abhängig sein, dass die Signale nur dann in Fahrtstellung gebracht werden können, wenn die Weichen für den Fahrweg richtig liegen und verschlossen sind (Signalabhängigkeit).“

Der funktionsfähige Zustand von Weichenanlagen ist essentiell für den Bahnbetrieb. Sind die personellen und finanziellen Ressourcen der Instandhaltung knapp bemessen, werden technische Mittel zur Zustandsüberwachung notwendig.
Die Herausforderung beim Betrieb eines Überwachungs- und Diagnosesystems liegt in der uneingeschränkten Funktionalität der Weiche, d.h. das Sensorbauteil darf keine Auswirkung auf die Aktivierung und den Ablauf des Weichenumlaufs haben. Nur wenige Systeme erfüllen diese Bedingung und liefern fernerhin zuverlässig akkurate Monitoringwerte [6]. Die Wahl fiel auf Grund der Möglichkeit einer einfachen Installation am Weichenmotor auf das Diagnosesystem SIDIS W, betrieben von der DB AG.
Das Deutsche Zentrum für Luft- und Raumfahrt (DLR) wurde damit beauftragt die gesammelten Daten zu analysieren, wobei sich Mitarbeiter des Instituts für Verkehrssystemtechnik dieser Aufgabe annahmen. Das Ziel der Untersuchung der gelieferten Stellstromdaten unter Abgleich mit Ausfalldaten war eine frühzeitige Erkennung von Störungen der Weichenanlagen, so dass etwaige Behinderungen des Eisenbahnverkehrs weitestgehend verhindert werden können.

Über einen Zeitraum von vier Monaten wurden im Bereich von sieben S-Bahnhöfen einer deutschen Großstadt Daten gesammelt, die anschließend von der DB AG an das DLR zur Aufbereitung und Auswertung übergeben wurden.
Der Einsatzzweck des Diagnosesystems bestand im Melden von Zustandsänderungen und Mängeln an Weichenkomponenten [28]. Der Umstand einer automatisierten technischen Überwachung eines Weichenobjekts bei jedem Weichenumlauf erlaubt jederzeit eine Aussage über den aktuellen Zustand der Weichenanlage, sodass der Betrieb des Diagnosesystems einen ersten Schritt in Richtung einer zustandsorientierten - weg von der fristenorientierten - Instandhaltung darstellt [28].
Das naheliegende wirtschaftliche Ziel war die Erhöhung der Verfügbarkeit jeder einzelnen Weiche und folgerichtig des gesamten Schienenverkehrnetzes. Langfristig erhofft sich die DB AG als Betreiber einen Effizienzgewinn der Weichenanlagen in der Bahninfrastruktur, was sich positiv auf die Pünktlichkeit der Züge auswirken würde.
Tabelle 2.0.1 listet konkrete Ziele, die sich im Zuge des Betriebs eines Diagnosesystems einstellen sollen.

- Zustände von Fahrweganlagen mit Messsystemen oder Diagnoseeinrichtungen überwachen.
- Veränderungen relevanter Parameter dieser Anlagen rechtzeitig erkennen.
- Zielgerichtete Maßnahmen zur Vermeidung von Störungen ergreifen, das heißt, eine vorausschauende Instandhaltung durchführen zu können.

Tabelle 2.0.1: Ziele der Überwachung und Diagnose [28]

2.1 Funktionsweise von SIDIS W

Das Diagnosesystem SIDIS W wird am Stellstromkabel der Eisenbahnweiche angebracht und überwacht lediglich eine Komponente dieser, den Weichenantriebsmotor, wobei der Stellstrom sowie die Stellspannung während des Weichenumlaufs aufgezeichnet wird.

Die Prozessabfolge des Weichenumlaufs erlaubt eine Unterteilung des Aufzeichnungsintervalls in verschiedene Phasen (Abbildung 2.1.1 links). Entsprechend können Mengen fortlaufender Wertepaare des Stellstroms und der -spannung (Wirkleistungskurve) den Etappen im Prozess zugeordnet werden. In der folgenden Tabelle 2.1.1 soll schrittweise die Datenerfassung des Systems und die anschließend stattfindende Aufbereitung der Daten erklärt werden,

während Tabelle 2.1.2 Kriterien listet, die einer Wirkleistungskurve entnommen werden können.

(1) Aufzeichnung von Stellstrom und -spannung

(2) Proportionalität zwischen abgegebener Kraft des Motors und der aufgenommenen Wirkleistung

(3) Übertrag in eine Wirkleistungskurve in Abhängigkeit von der Zeit

(4) Korrektur der Wirkleistung um den Einfluss der ohmschen Verluste auszugleichen

(5) Der weichentypische ohmsche Schleifenwiderstand wird ebenfalls in den Weichenstammdaten berücksichtigt

(6) Die berechnete Wirkleistungskurve der letzten zugeordneten Aufzeichnung entspricht dem aktuellen technischen Zustand der Weiche

(7) Ermittlung der Weichenkennwerte durch Auswertung von sechs Phasen der Wirkleistungskurve

(8) Überführung der Weichenkennwerte in aussagekräftige Parameter

(9) Prüfung der Parameter auf Einhaltung von Grenzwerten und visuelle Auswertung durch Ampelindikatoren

Tabelle 2.1.1: Funktionsweise von SIDIS W [28]

- Umlaufzeit (Unterspannung)
- Zungenvorspannung
- Verschlusspeak 'Anliegende Zunge'
- Schwergang (Zunge / Verschluss)
- Zungenaufschlag
- Verschlussüberdeckung
- Stromunsymmetrie (Motor, Kabel, Kontakte)

Tabelle 2.1.2: Verfügbare Kriterien nach Phasierung der Wirkleistungskurve [28]

Aus Tabelle 2.1.1 wird deutlich, dass die Wirkleistungskurve eine entscheidende Rolle einnimmt und vereinfacht als Signal verstanden werden kann, da sie die Informationsquelle für den aktuellen technischen Zustand eines Objekts darstellt. An dieser Stelle soll darauf hingewiesen werden, dass dieses Signal in keinem Zusammenhang mit den Signalanlagen der DB steht. Der Übergang vom Signal zum Merkmal erfolgt im Zuge der - in Punkt 7 und 8 aufgeführten - Auswertung der Phasen und der Überführung in Merkmale. Die Merkmale werden im weiteren Verlauf der Arbeit, insbesondere bei Durchführung mathematischer Operationen als Parameter und Variablen bezeichnet.

Im Anschluss an die Identifizierung und Messung der Weichenkennwerte war angedacht, dass das Diagnosesystem SIDIS W eine auf den gemessenen Werten beruhende Gefahrenanalyse durchführt. Hierfür wurden zunächst für sämtliche Parameter Schwellenwerte definiert. Wich der Verlauf der Wirkleistungskurve in einer Phase stark von den bekannten Standardwerten (Baseline data) ab, so dass ein Grenzwert über- respektive unterschritten wurde (Abbildung 2.1.1 rechts), so erfolgte eine Meldung. Diese Meldung indizierte durch einen farbigen Ampelindikator das Ausmaß der Abweichung.
Die Informationen wurden von der Instandhaltung ausgewertet (Abbildung 2.1.2) und im Zuge dessen wurden die betroffenen Weichenanlagen überprüft. Nach einer Erprobungsphase kamen zuständige Mitarbeiter der DB AG zum Ergebnis, dass kritische Meldungen häufig fehlerhaft sind und nur in sehr seltenen Fällen mit tatsächlichen Störungen an den Weichen einhergehen. Im Gegenzug wurden Störungen registriert, ohne dass die Ampelindikatoren im Vorlauf darauf hinwiesen. Folglich wurde von zentraler Stelle in der Instandhaltung entschieden, dass die Warnungen (*Alerts*) nicht weiter als Störungsindikator zu

werten sind und fanden im weiteren Verlauf des Beobachtungszeitraums keine Verwendung mehr.

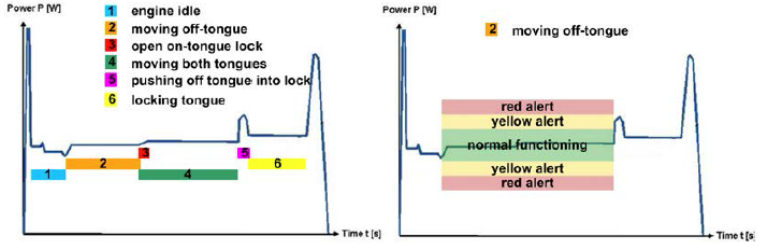

Abbildung 2.1.1: Phasen des Weichenumlaufs und Baseline data mit Grenzwertbereichen

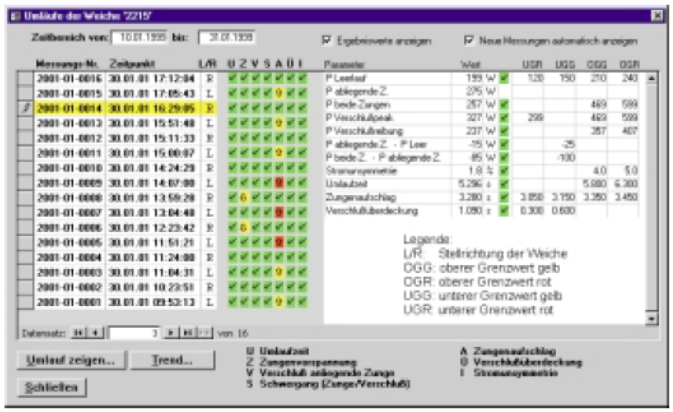

Abbildung 2.1.2: Monitoring der Alerts des Systems SIDIS W

2.2 Plant Asset Management

Das Plant Asset Management (PAM) hat seinen Ursprung im Asset Management (AM) und ist als aufbauendes, präzisierendes Element zu verstehen. Die Definition des Asset Managements umfasst drei Aspekte. Beginnend mit der strategischen Sicht, dem Verwalten der Assets über den gesamten Lebenszyklus hinweg, über die Organisation des Einsatzes und Zustanderhaltens der Assets bis hin zur konkreten Erzeugung und Bereitstellung von Informationen, insbesondere über Verlauf und Prognose der Asset-Gesundheit (Funktionstüch-

tigkeit) zur Entscheidungsunterstützung [29]. Der Zusatz *Plant* macht deutlich welche Art von Assets betrachtet wird, wodurch der Bogen geschlagen ist von allgemeinen Assets wie Vermögen, Brücken oder Fuhrparks zu den für die Prozessindustrie relevanten technischen Apparaturen. Entsprechende Assets können einer der drei Kategorien „Anlage“, „Teilanlage“ und „Anlagenteil“ zugeordnet werden.
Die Eingliederung des in dieser Arbeit betrachteten Objekts der Eisenbahnweiche erfolgt entsprechend seiner Position in der betrieblichen Struktur der DB AG. Weichen sind den beweglichen Fahrwegelementen zugehörig und können als zentraler Bestandteil dieser Organisationseinheit betrachtet werden. Dementsprechend werden die Objekte der Kategorie *Anlage* zugeordnet.

Eine Untersuchung der Anlage Bahnweiche im Sinne des PAM unterstützt das Verständnis dieser, indem die einzelnen Funktionen verkettet und in Bezug zueinander gesetzt werden, um ein strukturiertes Gesamtbild des Objekts zu gewinnen. Bezogen auf das in dieser Arbeit im Fokus stehende Weichendiagnosesystem SIDIS W ist von Interesse an welchen Stellen der Informationsaustausch gefördert und die im PAM verankerten Aufgaben der Weichenanlagen übernommen werden.

Des Weiteren beschränkt man sich innerhalb des Lebenzyklus eines Assets auf die Phasen Betrieb und Instandhaltung. Die Durchführung einer Diagnose ist Bestandteil der Instandhaltung, wobei die Schrittabfolge von SIDIS W (Tabelle 2.1.1) der Überwachungsfunktion der Instandhaltung (Tabelle 2.2.1) zugeteilt werden kann. Die Funktion *Überwachung* beginnt mit dem Übergang vom Signal zum Merkmal durch die Signalerfassung (Schritte (1)-(3)), gefolgt von der Signalaufbereitung (Schritte (4)-(7)) und der Merkmalserzeugung (Schritt (8)). Schritt (9) entspricht der anschließenden Merkmalsauswertung, die das Wissen, welches aus den gesammelten Informationen gewonnen wurde, in vordefinierte Symptome überführt.
In den Phasen der Diagnosefindung und Ursachenermittlung ist es notwendig den technischen Zustand einer Weiche durch Hinzunahme der Weichenstammdaten, um objektspezifische Produktmerkmale zu erweitern, da sämtliche Einflüsse, die das Symptom auslösen können, berücksichtigt werden müssen [29]. Weichenstammdaten können somit als Produktdaten verstanden werden, die eine grundlegende Datenbasis zur Identifizierung eines Assets und Zuordnung seiner Daten darstellen [29].
Die anschließende Analyse soll unter Abgleich mit den Störungsdaten Aufschluss über kritische Zustandsmerkmale geben, um mit Hilfe der gewonnenen Erkenntnisse eine Störungsfrüherkennung zu ermöglichen.

1. Überwachung	2. Diagnose	3. Prognose
1.1. Signal	2.1 Diagnosefindung	Voraussage
↓	↓	
1.2. Merkmale	2.2 Ursacheermittlung	
↓	↓	4. Therapie
1.3. Symptom	2.3 Bewertung	Maßnahmenvorschläge

Tabelle 2.2.1: Funktionenmodell der Instandhaltung [29]

Der Nutzungsaspekt der Instandhaltung sowie des Betriebs teilt dem Diagnosesystem seine zentrale Aufgabe zu, die Bereitstellung von Informationen. Nach der Gewinnung und Bereitstellung der Daten durch SIDIS W ist der folgende Informationsaustausch bzw. -fluss von Interesse, d.h. wie diese Daten weiterverarbeitet und die PAM-Funktionen der Instandhaltung und des Betriebs abgebildet werden. Um dies zu konkretisieren, werden die Kernpunkte Betriebsdaten, Ausfall + Ausfallvorhersage, Equipmenthistorie, Zustandserfassung und Kennzahlen (Instandhaltung) sowie Verfügbarkeitsvorhersage und Alarm/Analyse (Betrieb) genannt [29]. Basierend auf eine funktionierende Zustandserfassung, folgt die Zustandsbewertung eines Assets. Im Verbund bilden beide Funktionen das Condition Monitoring. Die Zustandsüberwachung stellt eine Schwerpunktaufgabe des PAM dar [29].
Während die Erfassung von Informationen in den Stellwerken erfolgt (siehe Tabelle 2.1.1), wird das Monitoring auf zentraler Ebene in der Instandhaltung durchgeführt. Da die Weichen im Verbund gemeinsam mit anderen Fahrwegelementen ein Netzwerk innerhalb der Bahninfrastruktur bilden, ist es notwendig Abhängigkeiten unter Objekten zu erkennen. Im gleichen Zuge müssen die Daten von SIDIS W, welche den technischen Zustand beschreiben mit Produktdaten in Form von Weichenstammdaten kombiniert werden. Auch analoge Informationen der Assets, welche beispielsweise bei einer Inspektion gewonnen wurden sind Teil des Gesamtzustands.

Das Gebiet der Störungsfrüherkennung ist von essentieller Bedeutung und an diesem Punkt ist ein Übergang von Betrieb zur Instandhaltung der Assets ersichtlich. So sollen kritische Objektzustände erkannt werden, wobei die Kondition eines Objekt dem aktuellen Verlauf der technischen Aufzeichnungen entspricht. Wird das Verhalten als kritisch eingestuft, muss dies selbstverständlich eine Überprüfung des Objekts nach sich ziehen, um die Funktionalität und Verfügbarkeit der Weiche sicherzustellen. Dabei bleibt festzuhalten, dass das Diagnosesystem SIDIS W selbst als Ursache für kritisches Verhalten ausgeschlossen und auch die aufgezeichneten Werte der Wirkleistungskurven als korrekt angenommen werden können.

In den folgenden Kapiteln dieser Arbeit soll zum Einen ein besonderer Augenmerk auf die Verknüpfung der technischen Zustandsparametern mit stationären

Stammdaten der Bahnweichen gelegt werden. Darauf beruhend sollen ähnliche Objekte in Gruppen zusammengefasst werden. Zum Anderen findet eine statistische Störungs- bzw. Ausfallvorhersage Anwendung.

Kapitel 3

Einführung der Objekte

Nachdem das PAM im vorangegangenen Kapitel 2.2 der Funktionsweise und dem Informationsfluss des Diagnosesystems SIDIS W eine ökonomische und betriebswirtschaftliche Bedeutung zugewiesen hat, bedarf es nun einer eindeutigen und einheitlichen Definition, Struktur sowie Organisation der beteiligten Faktoren, um eine Zustandserfassung sowie eine Cluster- und Störungsanalyse zu ermöglichen.

Ein Objekt ist definiert als (Daten-)Element mit spezifischer Semantik und Datenstruktur. Im Rahmen der Diplomarbeit sind die *beweglichen Fahrwegelemente Eisenbahnweichen* der DB AG als Objekte (W_m) zu verstehen. Im Beobachtungszeitraum von März bis Juni 2010 hat das SIDIS W Diagnosesystem von $M = 111$ Weichen verwertbare Daten aufgenommen.
Diese Objektmenge wird als Grundgesamtheit bezeichnet, da sämtliche enthaltenen Objekte gleichartig sind. Dabei begründet sich die Gleichartigkeit in einer einheitlichen Merkmalsmenge der Objekte [5]. Die Aufzeichnungszahl beträgt $N = 279619$, wobei der Umfang einer jeden einzelnen Weiche $N_m \in [7; 20519]\ \forall m = 1, \ldots, M$ stark variiert.

Im Vorlauf der Datenanalyse wurde sichergestellt, dass alle $M = 111$ Objekte mit einem funktionstüchtigem Diagnosesystem ausgestattet sind und somit über den gesamten Beobachtungszeitraum durchgängig überwacht werden. Etwaige fehlerhafte Aufzeichnungen bzw. Objekte, die sich durch fehlende Informationen oder ausschließlich fehlerhaften Daten auszeichnen wurden ausgeschlossen.

Ein vereinfachtes Zeitformat wurde durch Einführung eines Zeitstempels gewährleistet. Da das Verhalten der Bahnweichen vor dem Beobachtungszeitraum für die Berechnungen und Analyse unerheblich ist, wurde mit dem Beginn des Zeitraums die Zeitnahme gestartet und die Einheit Sekunde gewählt.

Es gilt:

$$\begin{aligned} &01.\ \text{März } 2010\ 00{:}00{:}00 \mathrel{\widehat{=}} 0 \\ &30.\ \text{Juni } 2010\ 23{:}59{:}59 \mathrel{\widehat{=}} 10540800 = T \end{aligned} \tag{3.0.1}$$

Unabhängig von der Datenerhebung des Diagnosesystems wurden $N^S = 75$ Störungen der Weichenobjekte registriert. Die Störungen konnten jeweils genau einem Objekt zugewiesen werden und die Zeitpunkte des Störungseintritts respektive -ende wurden aufgezeichnet.
Folglich kann eine Analyse der Daten, welche SIDIS W über den Beobachtungszeitraum sammelte unter Abgleich mit den Störungsdaten einer Ausfallvorhersage dienen (Kapitel 8).

3.1 Objektattribute und Parameter

Jede der N Aufzeichnungen, die per Definition jeweils einem Weichenumlauf oder einer Störung entsprechen beschreibt einen Datentupel mit einer festen Anzahl an Merkmalen und kann genau einem der M Objekte zugeordnet werden. Man spricht deshalb von der n-ten Aufzeichnung des Objekts W_m mit den Merkmalsausprägungen $\boldsymbol{X}_{m,n}$.
Die Merkmalsmenge, sowie das Verfahren der Merkmalserzeugung (Kapitel 2.1) ist selbstverständlich für alle Objekte gleich.

Die Attribute bzw. Parameter, welche verkettet den Datentupel bilden geben zum Einen die aktuell erzeugte Wirkleistungskurve und zum Anderen die statischen Produktdaten der Eisenbahnweichen wieder. Man unterscheidet deshalb zwei Attributklassen, die im weiteren Verlauf Messdaten und Weichenstammdaten genannt werden. Entsprechend entstehen folgende zwei Familien von Attributen:

$$\boldsymbol{A} = (A_1, \ldots, A_{15}) \quad \text{(Messdaten)} \tag{3.1.1}$$

$$\boldsymbol{B} = (B_1, \ldots, B_{15}) \quad \text{(Weichenstammdaten)} \tag{3.1.2}$$

$$x_{m,n,A_i} \ \ldots \text{ Ausprägung des Messdatenparameters } A_i \ldots \tag{3.1.3}$$

$$x_{m,n,B_i} \ \ldots \text{ Ausprägung des Produktdatenparameters } B_i \ldots \tag{3.1.4}$$

im n-ten Zustand der Weiche W_m

Für jedes Attribut definiert man einen Wertebereich, welcher alle angenommenen Werte des Attributs umfasst und die Semantik sowie den Datentyp be-

schreibt:

$$(i) \quad \forall A_i \text{ mit } i \in [0;15] \quad \exists\, DOM(A_i), \text{ sodass}$$
$$x_{m,n,A_i} \in DOM(A_i) \qquad \forall m = 1,\ldots,M;\ n \in [1,N_m].$$

$$(ii) \quad \forall B_i \text{ mit } i \in [0;15] \quad \exists\, DOM(B_i), \text{ sodass}$$
$$x_{m,n,B_i} \in DOM(B_i) \qquad \forall m = 1,\ldots,M;\ n \in [1,N_m]$$
$$x_{m,1,B_i} = x_{m,2,B_i} = \cdots = x_{m,N_m,B_i} =: x_{m,\cdot,B_i} \qquad \forall i \in [0;15].$$

Tabelle 3.1.1 ordnet den Attributen der Messdaten den Wertebereich in Form von Intervallen zu, während Tabelle 3.1.2 die Mächtigkeit des Wertebereichs für Attribute der Weichenstammdaten referiert.

Die Parameter der Messdaten $(A_1,\ldots,A_{15})$ werden in Tabelle 3.1.1 namentlich mit ihren Einheiten sowie den angesprochenen Wertebereichen gelistet und beschreiben gemeinsam die (technischen) Zustandsinformationen der Objekte. Die große Mehrheit von ihnen $(A_1,\ldots,A_{12})$ wird aus der Wirkleistungskurve im Prozess der Merkmalserzeugung (Kapitel 2.1) abgeleitet. Sämtliche relevante Kenngrößen des Weichenzustands werden durch die Parameter abgebildet, so dass auffällige Ausprägungen häufig den Vorgängen eines Weichenumlaufs zugeordnet werden können. Weiterhin ist die Berücksichtigung aller gelisteten Merkmale essentiell.

Neben den Merkmalen der Wirkleistungskurve gehören die Wetterdaten (A_{13} und A_{14}) der Liste an, um den Werteverlauf der Weichenmerkmale mit ihnen abgleichen zu können. Die Temperatur und die Niederschlagsmenge wurden von der nächstgelegenen Wetterstation eingeholt. Thomas Böhm hat in [6] erläutert, dass die Wetterdaten, speziell bei Unwettern mit hoher Niederschlagsmenge messbaren Einfluss auf die Wirkleistungskurve haben. Fernerhin ist der Zeitpunkt des Weichenumlaufs bzw. des Störungseintritts A_{15} festgehalten.

Die Messwerte zeichnen sich durch eine quantitative Datenstruktur aus, was bedeutet, dass eine Ordnung der Werte und mathematische Operationen zweier oder mehrerer Werte erlaubt sind.

Variable	Attributbezeichnung	Werteintervall	Einheit	Cluster-parameter
A_1	Pleer	$[55.4; 793.5]$	[W]	ja
A_2	Plauf1	$[61.3; 690.4]$	[W]	ja
A_3	Plauf2	$[84.4; 711.6]$	[W]	ja
A_4	Pverschl	$[0.0; 786.8]$	[W]	ja
A_5	PverschlPeakMax	$[4.17; 13.08]$	[W]	ja
A_6	Pzvorspg1	$[0.0; 85.9]$	[W]	ja
A_7	Pzvorspg2	$[-488.6; 278.7]$	[W]	ja
A_8	RelStellKraft	$[-404.2; 234.6]$	[W]	ja
A_9	Prutschmax	$[-0.2; 1097.1]$	[W]	ja
A_{10}	Tumlauf	$[360.8; 413.5]$	[W]	ja
A_{11}	Iunsym	$[0.0; 863.2]$	[W]	ja
A_{12}	Umotor	$[0.0; 197.0]$	[W]	ja
A_{13}	Temperatur	$[-4.1; 31.5]$	[°C]	nein
A_{14}	Niederschlag	$[0.0; 7.0]$	[mm]	nein
A_{15}	Aufzeichnungs-zeitpunkt	$[0; 10540800]$	[sec]	nein

Tabelle 3.1.1: Merkmale - Typ/Klasse Messwerte

Die Attributwerte der Weichenstammdaten $(B_1, \ldots, B_{15})$ (Tabelle 3.1.2) sind über den gesamten Beobachtungszeitraum konstant bzw. stationär. Diese Produktdaten werden im Gegensatz zu den dynamischen Bewegungsdaten der Messwerte nicht mit jeder Aufzeichnung aktualisiert, sondern verstehen sich als grundlegende Datenbasis zur Identifizierung und Konfiguration eines Assets sowie Zuordnung seiner Daten. Es handelt sich um bauartspezifische Informationen, sowie den ortsgebundenen Standort der Objekte.
Ein weiterer Unterschied zu den Attributen der Messwerte liegt in der nominalen Datenstruktur, welche keine Ordnung und nur eingeschränkt (mathematische) Operationen zulässt. Auch eine Substitution der Kategorien in eine quantitative, ordinale Struktur ist nicht möglich. Auf die daraus resultierenden Nachteile, die im Rahmen der Datenanalyse auftreten wird in Kapitel 5.4 der Clusteranalysemethoden detailliert eingegangen.

Variable	Attributbezeichnung	Ausprägungszahl	Cluster-parameter
B_1	Weichentyp	4	ja
B_2	Schienenprofil	3	ja
B_3	Radius der Abzweigung	6	ja
B_4	Schwellenart	4	ja
B_5	Antriebsart	3	ja
B_6	Verschlussart	13	ja
B_7	Verschlussartzusatz	2	ja
B_8	Zungenrollenpaare	11	ja
B_9	Endlagenprüfer	2	ja
B_{10}	Heizungsart	7	ja
B_{11}	Bahnhof	7	nein
B_{12}	Abzweigerichtung	3	nein
B_{13}	Abzweigewinkel	5	nein
B_{14}	Zungenbauart	1	nein
B_{15}	Zungenprüfkontakt	2	nein

Tabelle 3.1.2: Merkmale - Typ/Klasse Weichenstammdaten

3.2 Zustand eines Objekts

Der funktionale Status eines Objekts W_m zu einem Zeitpunkt $t \in [0; T]$ wird durch das Datentupel der n-ten Aufzeichnung $\boldsymbol{X}_{m,n}$ wiedergegeben, falls gilt $x_{m,n,A_{15}} \leq t < x_{m,n+1,A_{15}}$.

Dem liegt die Annahme zu Grunde, dass sich der Zustand eines Objekts zwischen zwei Aufzeichnungen nicht verändert, was bedeutet, dass der Zustandsverlauf eines jeden Objekts mit der ersten Aufzeichnung über den gesamten Beobachtungszeitraum hinweg bekannt ist. Die zurückliegenden Zustände werden gespeichert.

Durch Bilden des Datentupels einer Aufzeichnung $X_{m,n}$ werden die Wertemengen der Attribute, wie sie in den Tabellen 3.1.1 und 3.1.2 aufgeführt sind miteinander verknüpft und bilden gemeinsam den Zustandsraum. Die Zusammenstellung der Merkmale sowie der Wertebereich derselben ist selbstverständlich für alle Objekte identisch.
Die Verkettung qualitativer und quantitativer Daten führt zu einer hohen Komplexität des Zustandsraums. Daneben bleibt eine Unstimmigkeit in den Datenstrukturen der Parameter festzuhalten, da man dynamische Messwerte mit statischen Produktdaten kombiniert, was zur Folge hat, dass bei einer Aufzeichnung

lediglich der dynamische Teil aktualisiert werden muss. Diese beiden Besonderheiten müssen in den mathematischen Gebieten der statistischen Analyse und insbesondere der Clusteranalyse berücksichtigt werden.

Um eine Störung bzw. einen Ausfall eines Objekts im Zustandsraum zu indizieren, muss ein Störungsparameter angehängt werden. Das Phänomen einer Störung wird im folgenden Unterkapitel 3.3 thematisiert.

3.3 Störung eines Objekts

Im Beobachtungszeitraum $[0; T]$ wurden $N^S = 75$ Störungen aufgezeichnet. In Relation zu der Gesamtzahl von $N = 279619$ Aufzeichnungen erscheint dieser Wert sehr gering. Die Weichenumläufe dominieren entsprechend sehr stark die Aufzeichnungsdaten.
Eine Störung impliziert einen kritischen Zustands des Objekts, der sich durch eine eingeschränkte, nicht ordnungsgemäße Funktionstüchtigkeit des Objekts auszeichnet. Der Störungsfall ist nicht in jedem Fall mit einem Ausfall der Weiche gleichzusetzen, stattdessen kann es bei Unterlassen von Gegenmaßnahmen im weiteren Verlauf zu einem Ausfall kommen. Im Rahmen dieser Arbeit wird angenommen, dass Störungen der Weichenanlagen in direktem Zusammenhang mit dem technischen Verhalten im Vorlauf der Störung in Kombination mit den Produktdaten der Weichen stehen.
Die Anzahl der registrierten Störungen eines Objekts W_m, die sog. Störungszahl wird durch N_m^S beschrieben. Der Parameter bzw. das Merkmal $x_{m,n,S}$ indiziert den Störungszustand.

Die Störung eines Objekts stößt bei den Instandhaltern der DB AG einen Prozess an, der in den meisten Fällen ein Eingreifen erforderlich macht. Dies unterstreicht die meist schwerwiegenden Komplikationen, welche die Verfügbarkeit des Eisenbahnnetzes beeinträchtigen. Die Mitarbeiter der Instandhaltung, die mit der Reparatur des Objekts und der Bereitstellung der Funktionstüchtigkeit beauftragt wurden sind in der Datenquelle hinterlegt und registrierten zu jeder Störung eine Priorität, die einen Schweregrad derselben wiedergeben soll. Daneben wird, falls bekannt die Ursache der Störung festgehalten.

Eine Störung ist immer genau einem Objekt zugewiesen. Das vorhandene Weichennetz trägt dazu bei, dass die Störung einer Weichenanlage die Weichenumlaufsfrequenz einer oder mehrerer benachbarter Weichen beeinflussen kann, jedoch bleibt festzuhalten, dass die Zustände der Objekte paarweise unabhängig voneinander sind.
Für die in Kapitel 8 folgende Störungsanalyse gilt die Annahme, dass eine Störung auf den letzten Zustand bzw. den zurückliegenden Zustandsverlauf zurückgeführt werden kann. Dabei sorgt ein Störungsindikator innerhalb des

Zustandsraums für eine Abgrenzung dieser kritischen Zustände innerhalb der großen Aufzeichnungsmenge eines Objekts.
Dies stellt die Möglichkeit einer Störungsfrüherkennung in Aussicht, wie sie im Nutzungsaspekt der Instandhaltung (Kapitel 2.2) des PAM aufgeführt ist. Mit der Reparatur bzw. dem Zurücksetzen des Weichenobjektes sowie dem Sicherstellen der Funktionstüchtigkeit durch die Instandhalter (Störungsende) wird der kritische Zustand verlassen. Die Differenz zwischen den Zeitpunkten des Störungsende und -anfangs entspricht der Störungsdauer.

Kapitel 4

Statistische Größen

An dieser Stelle sollen die in Kapitel 3 beschriebenen Faktoren aufgegriffen und in statistische Größen überführt werden. Dabei werden stochastische Zusammenhänge herausgestellt, um eine anschließende statistische Störungsanalyse zu ermöglichen.

Eine Aufzeichnung entspricht einem Weichenumlauf oder einer Störung und kann genau einem Objekt zugeordnet werden. Entsprechend versteht man unter $N_m \in [7; 20519]$ $(m = 1, \ldots, M;\ M = 111)$ die Anzahl der Datenerhebungen eines Objekts.

Die Objektmenge

$$\{W_1, \ldots, W_M;\ M = 111\} \tag{4.0.1}$$

wurde in einem Zeitraum von März bis Juni 2010 durchgehend überwacht (3.0.1), so dass jede Aufzeichnung mit einem Zeitstempel

$$t \in [0; T], \quad T = 10540800$$

versehen wurde.

4.1 Attribute

Die Attribute bzw. Merkmale der Objekte wurden in den Tabellen 3.1.1 und 3.1.2 aufgelistet. Während sie im Kontext der Datenanalyse häufig als Parameter bezeichnet werden, spricht man in der Wahrscheinlichkeitstheorie von *(Zufalls-)Variablen*.

Für die Domains $DOM(A_i),\ i \in I^A$ und $DOM(B_i),\ i \in I^B$ gilt:

$$DOM(A_i) = \Omega_{A_i} \neq \emptyset \tag{4.1.1}$$

$$DOM(B_i) = \Omega_{B_i} \neq \emptyset \tag{4.1.2}$$

Die Mengensysteme $\xi_{A_i} \subset 2^{\Omega_{A_i}},\ i \in I^A$ und $\xi_{B_i} \subset 2^{\Omega_{B_i}},\ i \in I^B$ heißen $\sigma - \cap -$ *stabil* [13, S.1], da gilt:

(i) wenn für je abzählbar unendlich viele Mengen $a_1, a_2, \ldots \in \xi_{A_i}$ gilt,

$$\text{dann auch } \bigcup_{j=1}^{\infty} a_j \in \xi_{A_i} \quad \forall i \in I^A \tag{4.1.3}$$

(ii) wenn für je abzählbar unendlich viele Mengen $b_1, b_2, \ldots \in \xi_{B_i}$ gilt,

$$\text{dann auch } \bigcup_{j=1}^{\infty} b_j \in \xi_{B_i} \quad \forall i \in I^B \tag{4.1.4}$$

Man nutzt diese Eigenschaft, um zu zeigen, dass es sich bei den Mengensystemen um *Topologien* handelt. Folglich nennt man die Räume $\left(\Omega_{A_i}, \xi_{A_i}\right),\ i \in I^A$ und $\left(\Omega_{B_i}, \xi_{B_i}\right),\ i \in I^B$ topologisch [13, S.8f].

Die von den offenen Mengen erzeugten σ-Algebren

$$\mathscr{B}\left(\Omega_{A_i}\right) := \mathscr{B}\left(\Omega_{A_i}, \xi_{A_i}\right) := \sigma\left(\xi_{A_i}\right) \quad \forall i \in I^A \tag{4.1.5}$$

$$\mathscr{B}\left(\Omega_{B_i}\right) := \mathscr{B}\left(\Omega_{B_i}, \xi_{B_i}\right) := \sigma\left(\xi_{B_i}\right) \quad \forall i \in I^B \tag{4.1.6}$$

heißen Borel'sche σ-Algebren auf Ω_{A_i} respektive Ω_{B_i}.
Adaptiert an die Nomenklatur der Stochastik, erfolgt die Bezeichnung der σ-Algebren wie folgt:

$$(i) \quad \mathscr{A}_{A_i} = \sigma\left(\xi_{A_i}\right) \quad \forall i \in I^A$$

$$(ii) \quad \mathscr{A}_{B_i} = \sigma\left(\xi_{B_i}\right) \quad \forall i \in I^B$$

Die Messräume $\left(\Omega_{A_i}, \mathscr{A}_{A_i}\right),\ i \in I^A$ und $\left(\Omega_{B_i}, \mathscr{A}_{B_i}\right),\ i \in I^B$ heißen *polnisch*, da deren Topologien durch vollständige Metriken erzeugt werden [13, S.189].
Seien $\mathbb{P}_{A_i},\ i \in I^A$ bzw. $\mathbb{P}_{B_i},\ i \in I^B$ Maße auf den polnischen Räumen mit:

$$(i) \quad \mathbb{P}_{A_i}\left(\Omega_{A_i}\right) = 1 \quad \forall i \in I^A \tag{4.1.7}$$

$$(ii) \quad \mathbb{P}_{B_i}\left(\Omega_{B_i}\right) = 1 \quad \forall i \in I^B \tag{4.1.8}$$

dann heißen $\left(\Omega_{A_i}, \mathscr{A}_{A_i}, \mathbb{P}_{A_i}\right),\ i \in I^A$ und $\left(\Omega_{B_i}, \mathscr{A}_{B_i}, \mathbb{P}_{B_i}\right),\ i \in I^B$ *Wahrscheinlichkeitsräume*.

4.2 Störung

Die *Störungsvariable* $x_{m,n,S},\ m = 1, \ldots, M;\ n \in [1; N_m]$ kann im Fach der Stochastik als Zufallsvariable verstanden werden, die den Variablen aus Kapitel 4.1 gleichgestellt ist und mit ihnen den Zustand eines Objekts beschreibt.

Sie gibt wieder, ob es sich bei der n-ten Aufzeichnung des Objekts W_m um einen regulären Weichenumlauf oder eine Störung handelt und indiziert damit die Gattung der Aufzeichnung. Es gilt:

$$x_{m,n,S} = \begin{cases} 0 & \text{falls der Zustand des Objekts } W_m \\ & \text{zur } n\text{-ten Aufzeichnung unkritisch ist} \\ \Delta & \text{falls die } n\text{-te Aufzeichnung des Objekts } W_m \\ & \text{einer Störung entspricht} \end{cases} \tag{4.2.1}$$

$$x_{m,n,S} \in \Omega_S := \{0, \Delta\} \tag{4.2.2}$$

$$\mathscr{A}_S := \sigma(\Omega_S) \tag{4.2.3}$$

Die Störungszahl eines Objekts N_m^S hat folgende Eigenschaften:

$$(i) \quad N_m^S = |\{n \in [1; N_m] : x_{m,n,S} = \Delta\}| \tag{4.2.4}$$

$$(ii) \quad N_m^S \in [0; 7] \qquad \forall m = 1, \ldots, M \tag{4.2.5}$$

$$(iii) \quad \sum_{m=1}^{M} N_m^S = N^S \tag{4.2.6}$$

$$(iv) \quad |\{m : N_m^S \geq 1 \,;\, m = 1, \ldots, M\}| = 39. \tag{4.2.7}$$

Aus 4.2.7 folgt, dass 39 der 111 Objekte mindestens eine Störung aufweisen. Folglich funktionieren 72 Objekte über den gesamten Beobachtungszeitraum fehlerfrei.

4.3 Zustand

Der Zustand eines Objekts W_m wird nach jedem Weichenumlauf und mit jedem Störungseintritt erneuert. Dementsprechend definiert man nach der n-ten Aufzeichnung $n \in [1; N_m],\ n \in \mathbb{N}$ den 31-dimensionalen *Zufallsvektor* [14, S.88]:

$$\boldsymbol{X}_{m,n} = (x_{m,n,A_1}, \ldots, x_{m,n,A_{15}}, x_{m,n,B_1}, \ldots, x_{m,n,B_{15}}, x_{m,n,S}) \\ \forall m = 1, \ldots, M;\ n = 1, \ldots, N_m \tag{4.3.1}$$

Es gilt:

$$\begin{aligned}
&\text{Sei } i \in [1;15] = I^A \ : \\
&x_{m,n,A_i} \in DOM(A_i), \\
&x_{m,n,A_i} \text{ ist } \left(\Omega_{A_i}, \mathscr{A}_{A_i}\right)\text{-messbar} \qquad \forall m = 1,\ldots,M;\ n \in [1;N_m] \qquad (4.3.2)
\end{aligned}$$

$$\begin{aligned}
&\text{Sei } i \in [1;15] = I^B \ : \\
&x_{m,n,B_i} \in DOM(B_i), \\
&x_{m,n,B_i} \text{ ist } \left(\Omega_{B_i}, \mathscr{A}_{B_i}\right)\text{-messbar} \qquad (4.3.3) \\
&x_{m,1,B_i} = x_{m,2,B_i} = \cdots = x_{m,N_m,B_i} = x_{m,\cdot,B_i} \qquad \forall m = 1,\ldots,M \qquad (4.3.4)
\end{aligned}$$

$$\begin{aligned}
&x_{m,n,S} \in \Omega_S \\
&x_{m,n,S} \text{ ist } (\Omega_S, \mathscr{A}_S)\text{-messbar} \qquad (4.3.5)
\end{aligned}$$

Der Zustand $\boldsymbol{X}_{m,n}$ wird von Objekt W_m zum Zeitpunkt $t = x_{m,n,A_{15}}$ eingenommen und hält bis $t = x_{m,n+1,A_{15}} - \epsilon,\ \epsilon > 0$ an.

Aus der genannten stationären Eigenschaft der Weichenstammdatenattribute folgt:

$$\begin{aligned}
\boldsymbol{X}_{m,n} = (x_{m,n,A_1} \times \cdots \times x_{m,n,A_{15}} \times x_{m,\cdot,B_1} \times \cdots \times x_{m,\cdot,B_{15}} \times x_{m,n,S}) \\
\forall m = 1,\ldots,M;\ n = 1,\ldots,N_m \qquad (4.3.6)
\end{aligned}$$

Man definiert die Produkt-Sigma-Algebren $\bigotimes_{i\in I^A} \mathscr{A}_{A_i}$ und $\bigotimes_{i\in I^B} \mathscr{A}_{B_i}$ durch Projektion auf $\prod_{i\in I^A} \Omega_{A_i}$ respektive $\prod_{i\in I^B} \Omega_{B_i}$ [13, S.38,280]:

$$\begin{aligned}
(i) \quad & \bigotimes_{i\in I^A} \mathscr{A}_{A_i} := \mathscr{A}(pr_i : i \in I^A) \\
(ii) \quad & \bigotimes_{i\in I^B} \mathscr{A}_{B_i} := \mathscr{A}(pr_i : i \in I^B)
\end{aligned}$$

Äquivalent entstehen die Produkte der messbaren Räume $\left\{(\Omega_{A_i}, \mathscr{A}_{A_i})\right\}_{i\in I^A}$ und $\left\{(\Omega_{B_i}, \mathscr{A}_{B_i})\right\}_{i\in I^B}$:

$$(i) \quad \bigotimes_{i\in I^A}\left(\Omega_{A_i}, \mathscr{A}_{A_i}\right) := \left(\prod_{i\in I^A}\Omega_{A_i}, \bigotimes_{i\in I^A}\mathscr{A}_{A_i}\right)$$

$$(ii) \quad \bigotimes_{i\in I^B}\left(\Omega_{B_i}, \mathscr{A}_{B_i}\right) := \left(\prod_{i\in I^B}\Omega_{B_i}, \bigotimes_{i\in I^B}\mathscr{A}_{B_i}\right)$$

Die Vereinigung wird vereinfacht wie folgt dargestellt:

$$(iii) \quad (E, \mathscr{E}) := \left([\prod_{j\in\{A,B\}}\prod_{i\in I^j}\Omega_{j_i}] \times \Omega_S \,,\, [\bigotimes_{j\in\{A,B\}}(\bigotimes_{i\in I^j}\mathscr{A}_{j_i})] \otimes \mathscr{A}_S \right) \tag{4.3.7}$$

Dann gilt $\forall m = 1, \ldots, M;\ n \in [1; N_m]$:

$$\boldsymbol{X}_{m,n} \text{ ist } (E, \mathscr{E})\text{-messbar.} \tag{4.3.8}$$

Offensichtlich handelt es sich bei $(E, \mathscr{E})$ um einen polnischen (Zustands-) Raum mit Borel'scher σ-Algebra $\mathscr{E} := \mathscr{B}(E)$. Dann heißt $(E, \mathscr{E})$ *Borel'scher Raum* [13, S.189].

Die Sammlung sämtlicher Zustände und damit das Verhalten eines Objekts im Beobachtungszeitraum $[0; T]$ wird durch die Folge der Zufallsvariablen

$$\left(\boldsymbol{X}_{m,n}\right)_{n\in[1;N_m]} \tag{4.3.9}$$

beschrieben. Man spricht in diesem Kontext von der *Familie von Zufallsvariablen* [13, S.59], welche den kompletten Zustandsverlauf eines Objekts W_m repräsentiert:

$$\boldsymbol{X_m} = \left(\boldsymbol{X}_{m,n} : n \in [1; N_m]\right) \tag{4.3.10}$$

Um eine Folge von Zuständen bzw. den Zustandsverlauf eines Objekts genauer analysieren zu können, ist die Einführung einer aussagekräftigen statistischen Größe sinnvoll, die mehrere aufeinanderfolgende Zustände in sich vereint. Um dies zu gewährleisten, definiert man

$$\boldsymbol{X_m} \text{ als } \textit{stochastischen Prozess} \text{ von Zufallsvariablen bzw. -vektoren } \boldsymbol{X}_{m,n} \tag{4.3.11}$$

auf einem Wahrscheinlichkeitsraum $(E, \mathscr{F}, \mathbb{P})$ mit Aufzeichnungsbereich $[1; N_m] \subset \mathbb{N} \subset \mathbb{R}$ und Zustandsraum $(E, \mathscr{E})$ [13, S.193].

Eine Familie $\mathbb{F} = (\mathscr{E}_n; n \in [1; N_m])$ von σ-Algebren mit $\mathscr{E}_n \subset \mathscr{F}$ für jedes $n \in [1; N_m]$, heißt *Filtration* [13, S.195f], falls

$$\mathscr{E}_n := \sigma\left(\left\{\boldsymbol{X}_{m,n'}\right\}_{n'<n}\right) \qquad \forall n \in [1; N_m]$$
$$\mathscr{E}_{n-1} \subset \mathscr{F}_n \qquad \forall n \in [2; N_m]$$

$\mathbb{F}$ ist eine Folge monoton wachsender σ-Algebren $\mathscr{E}_n \subseteq \mathbb{F}$.
Die Filtration, als Familie von verschachtelten σ-Algebren heißt weiterhin *rechtsseitig stetig*, da gilt:

$$\forall n_1, n_2 \in [1; N_m]: \quad \bigcap_{n_2 > n_1} \mathscr{F}_{n_2} = \mathscr{F}_{n_1}$$

$\mathbb{F}$ wird als *natürliche Filtration* bezüglich der Folge $(\boldsymbol{X}_{m,n})_{n\in[1;N_m]}$ bezeichnet [25, S.428].

Im Rahmen der Diagnose von Bahnweichen bedeutet dies eine Zunahme der Informationsmenge mit fortschreitendem Beobachtungszeitraum aus der Sichtweise eines Objekts, d.h. mit jedem Weichenumlauf respektive Störungsvorfall wächst das Wissen über die Funktionsweise der Weiche.

Ein Stochastischer Prozess $\boldsymbol{X}_{\boldsymbol{m}}$ heißt *adaptiert* an die Filtration $\mathbb{F}$, falls $\boldsymbol{X}_{m,n}$ bezüglich $\mathscr{F}_n$ messbar ist für jedes $n \in [1; N_m]$. Gilt

$$\mathscr{F}_n = \sigma\left(\boldsymbol{X}_{m,n'}, n' \leq n\right) \qquad \forall n \in [1; N_m] \quad ,$$

so schreibt man $\mathbb{F} = \sigma(\boldsymbol{X}_{\boldsymbol{m}})$ die von $\boldsymbol{X}_{\boldsymbol{m}}$ erzeugte Filtration [13, S.195].

Kapitel 5

Clusteranalyse

Die Clusteranalyse gilt als Segmentierungsmethode des Data-Mining. Mit dem Begriff Data Mining wird die Anwendung geeigneter Methoden zur Entdeckung von Strukturen und Beziehungen in großen Datenmengen umschrieben [5, S.253]. Data-Mining ist elementarer Bestandteil der Wissenschaft des Knowledge Discovery in Databases (KDD). Diese umschreibt den gesamten Prozess der interaktiven und iterativen Entdeckung und Interpretation von nützlichem Wissen aus Daten [5, S.253].

Das Clustering ist eine weit verbreitete Technik, welche Daten(-elemente) auf Grundlage ihrer Ähnlichkeiten oder Unterschiede in Gruppen (genannt *Cluster*) partitioniert, sodass Elemente eines Clusters untereinander ein höheres Maß an Ähnlichkeit vorweisen, als zu Elementen anderer Cluster [20].

Es sollen im Folgenden geeignete Verfahren der Clusteranalyse vorgestellt werden, welche die Menge von $M = 111$ Objekten in eine feste Anzahl von k möglichst homogenen Gruppen unterteilt. Die Klassen bündeln dabei mehrere Objekte, die in ihren Eigenschaften, ihrem Verhalten, ihren Bauelementen, ihrer Herkunft und/oder ihres Einsatzortes übereinstimmen. Dies ist das Grundprinzip der Clusteranalyse.
Das Clustering verfolgt daher - im Sinne des Data-Minings - das Ziel eine verborgene Struktur bzw. ein Muster der Objektdaten aufzudecken [10].

Die Objektzahl $M = 111$ kann in der Disziplin des Clustering als verhältnismäßig gering beurteilt werden. In der Regel behandeln Anwendungen der Clusteranalyse mächtigere Grundgesamtheiten, wobei die Parameter eines Objekts durch einen eindimensionalen Vektor beschrieben werden. Die Herausforderung bei der Analyse der Instanz dieser Arbeit liegt jedoch in der Komplexität der Parametermenge, was bereits in Kapitel 3.1 dargelegt wurde. In Kapitel 4.1 wurden die Attribute anschließend in statistische Variablen überführt, um eine mathematische Basis für die Cluster- und Störungsanalyse zu schaffen.

Vor dem Hintergrund der Unterteilung der Attributmenge in zwei Klassen, erscheint eine Durchführung von zwei unabhängigen Verfahren sinnvoll, um anschließend die Ergebnisse zusammenzuführen (Kapitel 7).

5.1 Clusteranalyse Methoden

Den Gedanken des ersten Absatzes dieses Kapitels aufgreifend, kann man eine Optimallösung der Segmentierung wie folgt beschreiben: Objekte sollen derart in Klassen oder Gruppen zusammengefasst werden, dass zwischen den Elementen derselben Klasse größtmögliche Ähnlichkeit und zwischen den Elementen verschiedener Klassen größtmögliche Verschiedenheit besteht [5, S.173]. Um ein Maß für die Ähnlichkeit zweier Objekte zu erhalten, wurde der Distanzbegriff eingeführt, wobei der Abstand zweier Objekte durch einen Vergleich der Ausprägungen ihrer Parameter beschrieben wird. Es existieren zahlreiche Distanzmaße, die häufig für bestimmte Datenstrukturen der Objektparameter ausgelegt sind.
Weiterhin unterscheiden sich die Clusteringverfahren meist signifikant in den Vorgängen der initialen Clustererzeugung und des Clustersuchprozesses.
Im Zentrum eines Clusters kann ein fiktives Objekt, welches stellvertretend für die Merkmale sämtlicher inliegender Elemente steht generiert werden. Dieser *Clusterzentroid* zeichnet sich durch die minimale Summe der Distanzen zu den dem Cluster zugeordneten Objekten aus. Das Resultat eines Verfahrens bzw. einer Iteration ist durch die k Clusterzentroide und der Zuordnungstabelle für die Objekte eindeutig bestimmt.

Bei der Einordnung von Clusteranalysemethoden unterscheidet man auf oberster Ebene hierarchische und partitionierende Verfahren.
Partitionsmethoden zeichnen sich durch eine feste Clusterzahl k aus, welche die Anzahl der zu bildenden Gruppen beschreibt. Zu Beginn eines Verfahrens werden k Clusterzentroide gebildet, wobei zur Vereinfachung oftmals k Objekte zufällig gewählt werden, welche diese Rolle übernehmen [24]. Daraufhin werden die Objekte einzeln ihrem nächstgelegenen Zentroiden zugeordnet (*iterierte Minimaldistanzverfahren*). Alternativ können im Rahmen von *Austauschverfahren* Objektpaare unterschiedlicher Cluster getauscht werden, wenn sich ihre Distanzen zu den Clusterzentroiden in Summe verkleinern würden. Anschließend müssen im nächsten Schritt die Clusterzentroide entsprechend der neuen Objektzuordnung aktualisiert werden. Diese Schritte werden iterativ fortgeführt bis eine Stoppbedingung erfüllt ist.

Im Gegensatz zu den partitionierenden Verfahren wird bei den hierarchischen Methoden die Clusterzahl nicht mit dem Initialschritt festgelegt. Stattdessen wird sie während des Prozesses der Suche einer Optimallösung variiert. Hierarchische Verfahren können zwei Kategorien, die mit agglomerativ bzw. divisiv

definiert sind zugeordnet werden [24, 5]. *Divisive* Methoden starten im Initialschritt mit der Zusammenfassung sämtlicher Objekte zu einem Cluster, um anschließend iterativ die Gruppen zu splitten (*Top-Down*). Dabei erhöht sich mit jedem Schritt und der Abtrennung einer Objektgruppe oder eines einzelnen Objekts die Clusterzahl k um 1. *Agglomerativ* impliziert, dass im ersten Schritt jedes Objekt seinem eigenen Cluster zugeordnet wird, wobei selbstverständlich der Clusterzentroid mit dem jeweiligen Objekt übereinstimmt. In den darauf folgenden Schritten werden ähnliche Objekte gefunden und durch einen Zusammenschluss dieser ein neues Cluster gebildet (*Bottom-up*). Im agglomerativen Fall verringert jeder Zusammenschluss die Clusterzahl k um 1.

Im Folgenden soll zunächst erörtert werden welche Clusterzahl k der Instanz dieser Arbeit unter Berücksichtigung sämtlicher Größen und den betrieblichen Gegebenheiten in den Stellwerken und der Instandhaltung gerecht wird, um anschließend konkrete Verfahren der Clusteranalyse vorzustellen.

5.2 Wahl der der Clusterzahl

Mit der Entscheidung für eine Methode der Clusteranalyse geht die Auswahl der Clusterzahl einher. Es ist daher notwendig, Überlegungen über die Festsetzung der Größe k in den Entscheidungsprozess bei der Verfahrenswahl einfließen zu lassen.

Unter Berücksichtigung der Problemstellung dieser Arbeit sowie des existenten PAM für Weichenanlagen der DB AG scheint ein vorgegebenes Intervall für k sinnvoll.
Die Klassifikation der Weichenobjekte wirkt sich direkt auf die Organisationseinheit *bewegliche Fahrwegelemente* aus. Dabei sind Weichen nicht isoliert von anderen Fahrwegelementen, sondern lediglich als ein Faktor im Schienennetz zu betrachten. Deshalb gilt es die möglichen Gruppenmächtigkeiten in Betracht zu ziehen, um ein funktionierendes Monitoring und eine effiziente Instandhaltung zu ermöglichen.
Ein weiterer Gesichtspunkt ist die Diversität der Produktmerkmale. Die Stammdaten der Bahnweichen zeichnen sich durch eine geringe Mächtigkeit ihrer Wertemenge und eine quantitative Datenstruktur aus:

$$2 \leq |DOM(B_i)| \leq 13 \qquad \forall i \in I^B \tag{5.2.1}$$

Ein „großes“ k würde die Diversifizierung der Clusterzentroide in den Attributen der Weichenstammdaten enorm erschweren.
Folglich ist eine Justierung der Clustergröße (Mächtigkeit eines Cluster) durch Kontrolle der Clusterzahl k notwendig. Um trotz der genannten Einschränkun-

gen Optionen der Variation zu bieten, wurde das folgende Intervall gewählt:

$$k \in [3;10] \quad k \in \mathbb{N} \tag{5.2.2}$$

Folgende Bedingung für die Clusterzahl wird demnach bei den Clusterverfahren erfüllt:

$$k \leq M \tag{5.2.3}$$

Im Hinblick auf die Restriktion 5.2.2 fiel die Wahl auf die Klasse der Partitionsmethoden, welche die Suche nach Optimallösungen aufnehmen soll.
Vor der Anwendung expliziter Verfahren der Clusteranalyse ist es notwendig die Attribute für die Operationen vorzubereiten. Um einen Übergang der vordergründig hermeneutischen Attribute hin zu operativen Größen zu illustrieren wird terminologisch der Begriff *Variable* verwendet.

5.3 Vorbereitung der Variablen

In Kapitel 3.1 wurden die Objektattribute bereits eingeführt. Die Parameter der Messwerte und Weichenstammdaten müssen nun dahingehend bewertet werden, ob sie aussagekräftige Informationen über den Zustand der Objekte liefern. Das Ziel ist die Zusammenstellung einer Parametermenge, die jedes Objekt in seinen Eigenschaften und seinem Verhalten repräsentiert. Dies bedarf zum Einen einer Prüfung der aufgenommenen Daten auf Fehlerfreiheit und zum Anderen einer Eliminierung von Duplikaten.

Die kategorischen Produktdaten, bezogen auf die Weichenanlagen auch Weichenstammdaten genannt wurden in Tabelle (3.1.2) aufgelistet und parallel dazu wurden ihre Datenstruktur und -semantik herausgestellt. Die Tabelle liefert auch Aufschluss über die Verwendung der Attribute als Clusterparameter. Voraussetzung hierfür ist ein komponentenbeschreibender oder funktionsspezifischer Hintergrund der Variablen.
Entsprechend kann die Menge der Clustervariablen wie folgt definiert werden:

$$\begin{aligned} &\left\{B_i : \; i \in I^{B_c}\right\} \quad \text{mit} \\ &I^{B_c} := \{1,\ldots,10\}, \\ &I^{B_c} \subset I^B. \end{aligned} \tag{5.3.1}$$

Die Variablenwerte sind stationär und damit über den Beobachtungszeitraum konstant.

$$\begin{aligned} &b_{m,i} := x_{m,\cdot,B_i} \\ &b_{m,i} \in DOM(B_i) \qquad \forall m = 1,\ldots,M;\, i \in I^B \supset I^{B_c} \end{aligned} \tag{5.3.2}$$

Die größte Hürde beim Clustering von Objekten mit kategorischen Variablen liegt im Finden eines aussagekräftigen Distanzmaßes. Dieses Problem wird im folgenden Kapitel bei der Wahl eines geeigneten Clusteranalyseverfahrens thematisiert.

Um Variationen in den Clusterergebnissen zu erzeugen, werden Parameter- bzw. Variablentupel unterschiedlicher Länge ($n = 6, \ldots, 10$) generiert. Dies geschieht für jede Clusterzahl $k \in [3; 10]$, um später die Ergebnisse vergleichen und eine Bewertung der Variablenkombinationen vornehmen zu können.

$$V^B = \left\{ \binom{|I^{B_c}|}{n} : \; n = 6, \ldots, 10 \right\} \tag{5.3.3}$$

Auch die Menge der Messwertattribute $\boldsymbol{A} = (A_1, \ldots, A_{15})$ (Tabelle 3.1.1) muss auf für die Clusteranalyse relevante Variablen reduziert werden.

$$\left\{ A_i : \; i \in I^{A_c};\; I^{A_c} = \{1, \ldots, 12\},\; I^{A_c} \subset I^A \right\} \tag{5.3.4}$$

Da die Ausprägungen dieser Variablen nicht stationär sind, sondern während des Beobachtungszeitraums variieren, müssen die Werte im Vorlauf des Clusteringverfahrens aggregiert werden. So entsteht der gewünschte eindimensionale Variablenvektor für alle Objekte, welcher das technische Verhalten der Weichenanlagen charakterisiert.

Bei der Clusteranalyse bleiben Störungszustände unberücksichtigt, sodass nur die Informationen der Weichenumläufe ausgewertet werden. Im Zuge der Umstellung einer Weiche können beachtliche Unterschiede der Wirkleistung je nach Endlage (Links- bzw. Rechtslage) registriert werden. Dies hat zur Folge, dass für zahlreiche Parameter der Messwerte eine Abstufung sinnvoll ist. Für folgende Parameter wurden daher zwei Variablen eingeführt, wobei sichergestellt wurde, dass für jeden Objektzustand $X_{m,n}$ die Endlage bzw. Richtung des Weichenumlaufs $X_{m,n,\text{Richtung}}$ korrekt erfasst wurde.

$$\left\{ A_i : \; i \in \{1, \ldots, 9\},\; i \in I^{A_c} \right\} \tag{5.3.5}$$

Die Entscheidung bei der Wahl der Aggregationsmethode fiel auf den Mittelwert. Es werden insgesamt 21 Clustervariablen gebildet.

$$a_{m,i,r} := \frac{1}{N_m} \sum_{n=0}^{N_m} I(X_{m,n,\text{Richtung}} = r)\, x_{m,n,A_i}$$
$$\forall m = 1,\ldots,M;\ r \in \{L,R\};\ i \in I^{A_c} \cap \{1,\ldots,9\} \tag{5.3.6}$$

und

$$a_{m,i,0} := \frac{1}{N_m} \sum_{n=0}^{N_m} x_{m,n,A_i} \qquad \forall m = 1,\ldots,M;\ i \in I^{A_c} \setminus \{1,\ldots,9\} \tag{5.3.7}$$

Analog zu den Weichenstammdaten, werden auch für Messdaten Variablentupel gebildet. Sie haben eine Mächtigkeit von mindestens 16, höchstens 21.

$$V^A = \left\{ \binom{|\{a_{m,i,r} : i,r \in I^{A_c} \times \{0,L,R\}\}|}{n} : n = \{16,\ldots,21\} \right\} \tag{5.3.8}$$

Zusammenfassend beschreiben 21 Variablen das technische Verhalten und 10 Variablen die baulichen Eigenschaften sowie den Einsatzort der $M = 111$ Objekte. Die objektcharakterisierenden Clustervariablen unterscheiden sich nicht nur in ihrem Gehalt, sondern nach dem gleichen Muster auch in ihrer Datensemantik.

5.4 Clustering Weichenstammdaten

Die Clusterzahl k wurde in Kapitel 5.2 festgelegt. Es gilt $k \in [3;10]$, $k \in \mathbb{N}$ (5.2.2). Weiterhin wurden im vorangehenden Kapitel die Variablen bereits vorbereitet. Die Clusteranalyse erfolgt über die Kategorien der 10 Variablen $b_{m,i},\ i \in I^{B_c}$ (5.3.2), wobei Variablentupel der Mächtigkeit $n = 6,\ldots,10$ (5.3.3) gebildet werden.

Auf Grund der qualitativen Datenstruktur der Clustervariablen wurde das Clusteranalyseverfahren k-Modes gewählt, auf dessen Algorithmus und der Vorgehensweise bei der Suche nach einer Optimallösung im Folgenden näher eingegangen werden soll.

Die Variablen B_i lassen eine repräsentative Umwandlung in eine ordinale oder quantitative Datenstruktur nicht zu. Entsprechend ist es nicht möglich ein „feines“ und aussagekräftiges Distanzmaß wie beispielsweise die Differenz zu berechnen, um eine Aussage über Nähe oder Ähnlichkeit zweier Objekte zu erhalten [12]. Man muss sich beim Bestimmen eines Verhältnisses zwischen zwei Werten $b_{m,i}, b_{m',i} \in DOM(B_i)$ auf die Gleichheit bzw. Ungleichheit beschränken.

Übertragen auf die Ähnlichkeit zweier Objekten, wird die relative Anzahl übereinstimmender Variablen berechnet und als Maß verwendet (*Simple Matching Coefficient*) [24].

Es wird die zu minimierende Zielfunktion $P(Q, \boldsymbol{C})$ betrachtet [12].

$$P(Q, \boldsymbol{C}) = \sum_{l=1}^{k} \sum_{m=1}^{M} q_{m,l}\, d(W_m, C_l) \tag{5.4.1}$$

$$\text{u.d.N.} \quad \sum_{l=1}^{k} q_{m,l} = 1, \quad 1 \le m \le M \tag{5.4.2}$$

$$\sum_{m=1}^{M} q_{m,l} \ge 1, \quad 1 \le l \le k$$

$$q_{m,l} \in \{0,1\}, \quad 1 \le m \le M,\ 1 \le l \le k$$

$\boldsymbol{C} = (C_1, \ldots, C_k)$ sei die Familie der k Clusterzentroiden, mit:

$$C_l = \left(c_{l,i} :\ i \in I^{B_c},\ c_{l,i} \in DOM(B_i)\right) \qquad \forall l = 1, \ldots, k \tag{5.4.3}$$

Für das einfache Ungleichheitsmaß $\delta(b_{m,i}, b_{m',i})$, sowie das Distanzmaß zwischen zwei Objekten $d(W_m, W_{m'})$, betreffend einer Kombination von Variablen ihrer Weichenstammdaten aus V^B gilt:

$$\delta(b_{m,i}, b_{m',i}) := \begin{cases} 0 & \text{für } (b_{m,i} = b_{m',i}) \\ 1 & \text{für } (b_{m,i} \ne b_{m',i}) \end{cases} \qquad \forall\, b_{m,i}, b_{m',i} \in DOM(B_i),\ i \in I^{B_c} \tag{5.4.4}$$

$$d(W_m, W_{m'}) := \sum_{i \in I^{B_c}} \delta(b_{m,i}, b_{m',i}) \qquad m, m' \in \{1, \ldots, M\} \tag{5.4.5}$$

Äquivalent kann man die Ungleichheit zwischen Objekt und Clusterzentroid ebenfalls mit dem Simple Matching Coefficient bestimmen:

$$d(W_m, C_l) = \sum_{i \in I^{B_c}} \delta(b_{m,i}, c_{l,i}) \qquad m \in \{1, \ldots, M\},\ l = 1, \ldots, k \tag{5.4.6}$$

Q sei eine Zugehörigkeitsmatrix der Dimension $M \times k$ mit den Einträgen $q_{m,l} \in Q$:

$$q_{m,l} = \begin{cases} 1 & \text{falls } d(W_m, C_l) \le d(W_m, C_{l'}) \quad \forall l' = 1, \ldots, k \\ 0 & \text{sonst} \end{cases} \tag{5.4.7}$$

Man unterteilt das Minimierungsproblem (5.4.1) in zwei Unterprobleme:

(i) Problem P_1: Setze $\boldsymbol{C} = \hat{\boldsymbol{C}}$ und löse das Problem $P(Q, \hat{\boldsymbol{C}})$

(ii) Problem P_2: Setze $Q = \hat{Q}$ und löse das Problem $P(\hat{Q}, \boldsymbol{C})$

Problem P_1 wird gelöst, indem $\forall m = 1, \ldots, M$ die Einträge der Matrix $q_{m,l} \in Q$ neu berechnet werden. Daraus resultiert eine neue Zugehörigkeitsmatrix Q.
Problem P_2 löst man durch Ermittlung der Kategorien mit den maximalen relativen Häufigkeiten innerhalb eines Clusters für alle Clusterzentroiden. Die Zentroiden werden anschließend aktualisiert.

An dieser Stelle wird die relative Häufigkeit der Variablenkategorien eines Clusters eingeführt:

$$fr(B_i = b_{\tilde{m},i}|C_l, Q) := \frac{\sum_{m=1}^{M} q_{m,l}\,(1 - \delta(b_{m,i}, b_{\tilde{m},i}))}{\sum_{m=1}^{M} q_{m,l}}$$
$$\forall\, \tilde{m} = 1, \ldots, M \text{ mit } q_{\tilde{m},l} = 1;\ l = 1, \ldots, k \qquad (5.4.8)$$

Zur Lösung des Unterproblems P_2 ist es notwendig, dass für sämtliche Kategorien eines Clusters die dazugehörige Häufigkeit berechnet wird, um abschließend diejenige Kategorie zu wählen, welche die maximale relative Häufigkeit aufweist.

$$\begin{aligned} &\exists\, \hat{m} \in [1; M]: \\ &fr(B_i = b_{\hat{m},i}|C_l, Q) \geq fr(B_i = b_{m,i}|C_l, Q) \\ &\qquad q_{\hat{m},l}, q_{m,l} = 1,\ m = 1, \ldots, M \end{aligned} \qquad (5.4.9)$$

$$c_{l,i} = b_{\hat{m},i} \qquad \forall i \in I^{B_c} \qquad (5.4.10)$$

Zusammenfassend wird für alle $l = 1, \ldots, k$ ein neuer Clusterzentroid C_l definiert:

$$C_l = \left(c_{l,i} :\ i \in I^{B_c};\ c_{l,i} \in DOM(B_i)\right) \qquad (5.4.11)$$

Die Suche nach einem optimalen Resultat des Optimierungsproblems erfolgt iterativ und wird im folgenden Kapitel 5.5 anhand des verwandten k-Means Verfahrens erläutert. Ist die maximale Anzahl an Iterationen durchlaufen bzw. wurde das Verfahren abgebrochen, so wird die temporäre Matrix $Q = Q^{t_{\max}}$ mit

der Zugehörigkeitsmatrix $S^{M\times k}$ gleichgesetzt. Für die Einträge $s_{m,l} \in S$ gilt:

$$(i) \quad s_{m,l} = q_{m,l} \in \{0,1\} \qquad \forall\ 1 \le m \le M,\ 1 \le l \le k \tag{5.4.12}$$

$$(ii) \quad s_{m,l} = \begin{cases} 1 & \text{falls Weiche/Objekt } W_m \text{ in Weichenstammdaten-Cluster } l \\ 0 & \text{sonst} \end{cases} \tag{5.4.13}$$

$$(iii) \quad \sum_{m=1}^{M} s_{m,l} \ge 1 \quad \forall l = 1,\dots,k \tag{5.4.14}$$

$$(iv) \quad \sum_{l=1}^{k} s_{m,l} = 1 \quad \forall m = 1,\dots,M \tag{5.4.15}$$

$$(v) \quad \sum_{m=1}^{M}\sum_{l=1}^{k} s_{m,l} = M \tag{5.4.16}$$

5.5 Clustering nach Messwerte

Bei der Clusteranalyse für Messdaten sei die Clusterzahl k weiterhin unverändert im Intervall $[3;10]$ vorgegeben. Die Analyse wird über die durch Aggregation erhaltenen Werte der 21 Clustervariablen $a_{m,i,\cdot}$ durchgeführt, wobei die Zusammenstellung der Variablenmenge entsprechend V_m^A (5.3.8) je Durchgang variiert wird. Die Variablentupel haben dabei eine Länge von 16 bis 21. Nach Abschluss der Analyse soll dies Aufschluss über „gute“ bzw. „schlechte“ Variablenkombinationen geben.

Die k-Means Clusteringmethode zeichnet folgende Eigenschaften aus:

- k-Means besticht durch seine Effizienz bei großen Datensätzen.
- Die gefundene Optimallösung beschreibt häufig ein lokales Optimum [27].
- Das Verfahren arbeitet nur mit numerischen Werten.
- Die Cluster haben eine konvexe Form.

Tabelle 5.5.1: Eigenschaften des k-Means Verfahrens [12]

Im Folgenden soll der Algorithmus der verwendeten Clusteranalysemethode erläutert werden.

Man betrachte die zu minimierende Zielfunktion [12],[27]:

$$P(Q, \boldsymbol{C}) = \sum_{l=1}^{k} \sum_{m=1}^{M} q_{m,l}\, d(W_m, C_l) \tag{5.5.1}$$

$$\begin{aligned} \text{u.d.N.} \quad & \sum_{l=1}^{k} q_{m,l} = 1, \quad 1 \le m \le M \\ & \sum_{m=1}^{M} q_{m,l} \ge 1, \quad 1 \le l \le k \\ & q_{m,l} \in \{0,1\}, \quad 1 \le m \le M, 1 \le l \le k \end{aligned} \tag{5.5.2}$$

Wir schreiben $d(W_m, W_{m'})$ für die euklidische Distanz zwischen zwei Objekten W_m und $W_{m'}$ in den Variablen eines Tupels aus V^A:

$$\begin{aligned} d(W_m, W_{m'}) = & \sum_{i \in I^{A_c} \cap \{1,\ldots,9\}} \sum_{r \in \{L,R\}} \left(a_{m,i,r} - a_{m',i,r}\right)^2 \\ & + \sum_{i \in I^{A_c} \setminus \{1,\ldots,9\}} \left(a_{m,i,0} - a_{m',i,0}\right)^2 \qquad \forall m, m' = 1, \ldots, M \end{aligned} \tag{5.5.3}$$

Für die Zugehörigkeitsmatrix Q der Dimension $M \times k$ mit Einträgen $q_{m,l} \in Q$ gilt:

$$q_{m,l} = \begin{cases} 1 & \text{falls } d(W_m, C_l) \le d(W_m, C_{l'}) \quad \forall l' = 1, \ldots, k \\ 0 & \text{sonst} \end{cases} \tag{5.5.4}$$

$\boldsymbol{C} = (C_1, \ldots, C_k)$ sei die Familie der k Zentroiden, mit:

$$C_l = \left(c_{l,i,r} : \; i, r \in I^{A_c} \times \{0, L, R\}, \; c_{l,i,r} \in DOM(A_i)\right) \qquad \forall l = 1, \ldots, k \tag{5.5.5}$$

Das Minimierungsproblem (5.5.1) soll analog zu dem k-Modes Verfahren in zwei Unterprobleme unterteilt werden:

(i) Problem P_1: Setze $\boldsymbol{C} = \hat{\boldsymbol{C}}$ und löse das Problem $P(Q, \hat{\boldsymbol{C}})$

(ii) Problem P_2: Setze $Q <= \hat{Q}$ und löse das Problem $P(\hat{Q}, \boldsymbol{C})$

Problem P_1 wird gelöst, indem für alle $m = 1, \ldots, M$ die Einträge der Matrix $q_{m,l} \in Q$ neu berechnet werden. Dabei wird jeweils die kürzeste Entfernung zu einem Clusterzentroiden gesucht. Schlussendlich resultiert eine neue Zugehörigkeitsmatrix Q.
Problem P_2 löst man durch Berechnung der durchschnittlichen Variablenwerte

sämtlicher dem Cluster l zugehörigen Cluster:

$$c_{l,i,r} = \frac{\sum_{m=1}^{M} q_{m,l}\, a_{m,i,r}}{\sum_{m=1}^{M} q_{m,l}} \qquad \forall i, r \in I^{A_c} \times \{0, L, R\},\ l = 1, \ldots, k \tag{5.5.6}$$

Folglich erhält man einen neuen Clusterzentroiden und die Distanz zwischen diesem und einem Objekt lässt sich wie folgt berechnen:

$$C_l = \left(c_{l,i,r} : \ i, r \in I^{A_c} \times \{0, L, R\},\ c_{l,i,r} \in DOM(A_i)\right) \tag{5.5.7}$$

$$\begin{aligned} d(W_m, C_l) = & \sum_{i \in I^{A_c} \cap \{1,\ldots,9\}} \ \sum_{r \in \{L,R\}} \left(a_{m,i,r} - c_{l,i,r}\right)^2 \\ & + \sum_{i \in I^{A_c} \setminus \{1,\ldots,9\}} \left(a_{m,i,0} - c_{l,i,0}\right)^2 \qquad \forall m = 1, \ldots, M;\ l = 1, \ldots, k \end{aligned} \tag{5.5.8}$$

Das iterative Lösen dieser beiden Unterprobleme P_1 und P_2 führt zu einem lokalen oder globalen Minimum der Zielfunktion $P(Q, \boldsymbol{C})$. Im Folgenden soll der Initialschritt und die anschließende Schrittabfolge erläutert werden. Dabei verfahren die Algorithmen des k-Modes (Kapitel 5.4) und k-Means identisch.

1. Finde/Definiere ein erstes $\boldsymbol{C}^0$ und löse das Unterproblem P_1 $(P(Q, \boldsymbol{C}^0))$, um Q^0 zu erhalten. Setze $t = 0$.
   Initial werden k Objekte zufällig ausgewählt und als Clusterzentroide festgelegt. Dabei wird sichergestellt, dass für alle Variablentupel keine identischen Objekte existieren.

$$\begin{aligned} & \forall l = 1, \ldots, k \quad \exists \text{ genau ein } m = 1, \ldots, M : \\ & C_l = (c_{m,i,r} : \ i, r \in I^{A_c} \times \{0, L, R\},\ c_{m,i,r} \in DOM(A_i)) \\ & \qquad \text{mit } C_l \neq C_{l'} \qquad \forall\, l, l' = 1, \ldots, k;\ l \neq l' \end{aligned} \tag{5.5.9}$$

   bzw. im Falle von k-Modes:

$$\begin{aligned} & C_l = (c_{l,i} : \ i \in I^{A_c},\ c_{l,i} \in DOM(B_i)) \\ & \qquad \text{mit } C_l \neq C_{l'} \qquad \forall\, l, l' = 1, \ldots, k;\ l \neq l' \end{aligned} \tag{5.5.10}$$

2. Setze $\hat{Q} = Q^t$ und löse $P(\hat{Q}, \boldsymbol{C})$. Vergleiche die Werte der Zielfunktion und verwende als Parameter $\boldsymbol{C}^{t+1}$.

$$P(\hat{Q}, \boldsymbol{C}^{t+1}) - P(\hat{Q}, \boldsymbol{C}^t) = \begin{cases} 0 & \text{stoppe} \\ <> 0 & \text{führe fort mit Schritt 3} \end{cases} \tag{5.5.11}$$

3. Setze $\hat{\boldsymbol{C}} = \boldsymbol{C}^{t+1}$ und löse $P(Q, \hat{\boldsymbol{C}})$, um Q^{t+1} zu erhalten. Vergleiche erneut die Werte der Zielfunktion vor und nach diesem Schritt:

$$P(Q^{t+1}, \hat{\boldsymbol{C}}) - P(Q^t, \hat{\boldsymbol{C}}) = \begin{cases} 0 & \text{stoppe} \\ <> 0 & \text{setze } t = t+1 \text{ und gehe zu Schritt 2.} \end{cases}$$

Als weitere Stoppbedingung wird die maximale Zahl der Iterationen t begrenzt.

Nach Abschluss des k-Means Verfahrens wird die temporäre Matrix $Q = Q^{t_{\max}}$ in die Zugehörigkeitsmatrix $R^{M \times k}$ mit binären Einträgen $r_{m,l}$ überführt, wobei gilt:

$$(i) \quad r_{m,l} = q_{m,l} \in \{0,1\} \qquad \forall\, 1 \leq m \leq M,\, 1 \leq l \leq k \tag{5.5.12}$$

$$(ii) \quad r_{m,l} = \begin{cases} 1 & \text{falls Weiche/Objekt } W_m \text{ in Messdaten-Cluster } l \\ 0 & \text{sonst} \end{cases} \tag{5.5.13}$$

$$(iii) \quad \sum_{m=1}^{M} r_{m,l} \geq 1 \quad \forall l = 1, \ldots, k \tag{5.5.14}$$

$$(iv) \quad \sum_{l=1}^{k} r_{m,l} = 1 \quad \forall m = 1, \ldots, M \tag{5.5.15}$$

$$(v) \quad \sum_{m=1}^{M} \sum_{l=1}^{k} r_{m,l} = M \tag{5.5.16}$$

An dieser Stelle sollen die Modifikationen der k-Means Methode herausgestellt werden, die notwendig sind, um das Verfahren auf die Verwendung für kategorische Variablen (k-Modes) zu adaptieren.

- Verwenden eines einfachen Simple Matching Ungleichheitsmaß für kategorische Variablen.
- Ersetzen der Mittelwerte für Cluster durch Modi (Clusterzentroide).
- Anwenden einer häufigkeitsbasierten Methode für das Finden der Modi, um Unterproblem P_2 zu lösen.

Tabelle 5.5.2: Adaption von k-Means auf k-Modes [12]

5.6 Umsetzung in KNIME

Die Clusteranalyse in KNIME erfordert komplexe Workflows mit vielfältigen Modulstrukturen. Prinzipiell ist das Data-Mining-Tool für die Umsetzung eines Clusteringverfahrens vorbereitet, doch die Integration von einigen Methoden bedarf der Entwicklung von Zusatzmodulen. So ist der *k*-Modes Algorithmus beispielsweise im Gegensatz zu *k*-Means nicht Teil des Standardrepositories.
Die große Zahl an Variablentupel und die 8 Optionen bei der Wahl der Clusterzahl führen zu zahlreichen Durchläufen des Algorithmuskreislaufs (Tabelle 5.6.1), was einen enormen Rechenaufwand zur Folge hat. Der Kreislauf von sich wiederholenden Routen wird durch sogenannte Loop-Knoten realisiert (Abb. 5.6.1). So wird iterativ für jede Variabelkombination und für jede Clusterzahl k an Hand der Parameterwerte ein Clusterergebnis herbeigeführt und gespeichert.

Beispielhaft soll der Workflow, welcher die Ergebnisse der Clusteranalyse nach *k*-Means ausgibt das Zusammenspiel von Variationen in Clusterzahl und Parametermenge illustrieren (Abb. 5.6.1). Die Modifikationen des *k*-Means Algorithmus im Zuge der Clusteranalyse nach Weichenstammdaten (5.5.2) wurden komplett im *k*-Modes Knoten abgebildet, weshalb die Routen bis auf ein Modul identisch sind.

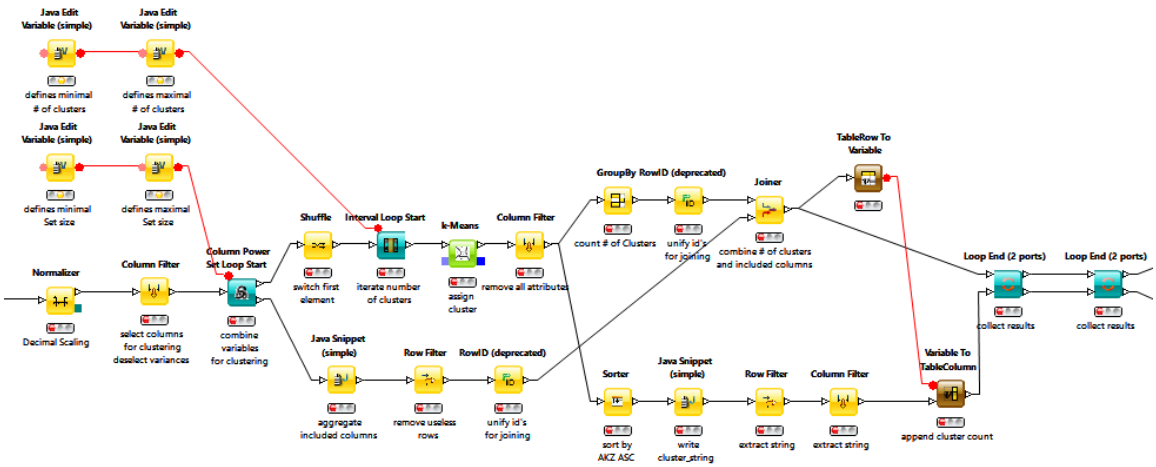

Abbildung 5.6.1: k-Means Workflow - Kreislauf des Algorithmus [KNIME]

Clusteringverfahren	Variablentupel	Clusterzahlen	gesamt
k-Modes	386	8	3088
k-Means	27915	8	233320

Tabelle 5.6.1: Anzahl der Algorithmen-Durchläufe und Clusterergebnisse

Die Abbildungen 5.6.2 und 5.6.3 zeigen das Konfigurationsmenü des *k*-Modes - sowie die Output-Tabellen des Tupelbildungsknoten. Das Modul, welches die Operationen des *k*-Modes Algorithmus ausführt wurde eigens im Zuge der Datenanalyse entwickelt. Auf Grund der Ähnlichkeit zum *k*-Means Algorithmus diente das Schema dieses Knotens aus dem Standardrepository als Vorlage. Um eine weiterführende Analyse der Clusterergebnisse im Rahmen eines Workflows zu gewährleisten, ist eine ressourcensparende Zwischenspeicherung der Ergebnisse notwendig. Es genügt an dieser Stelle die Clusterzugehörigkeiten der Objekte als verketteten String zu übergeben.

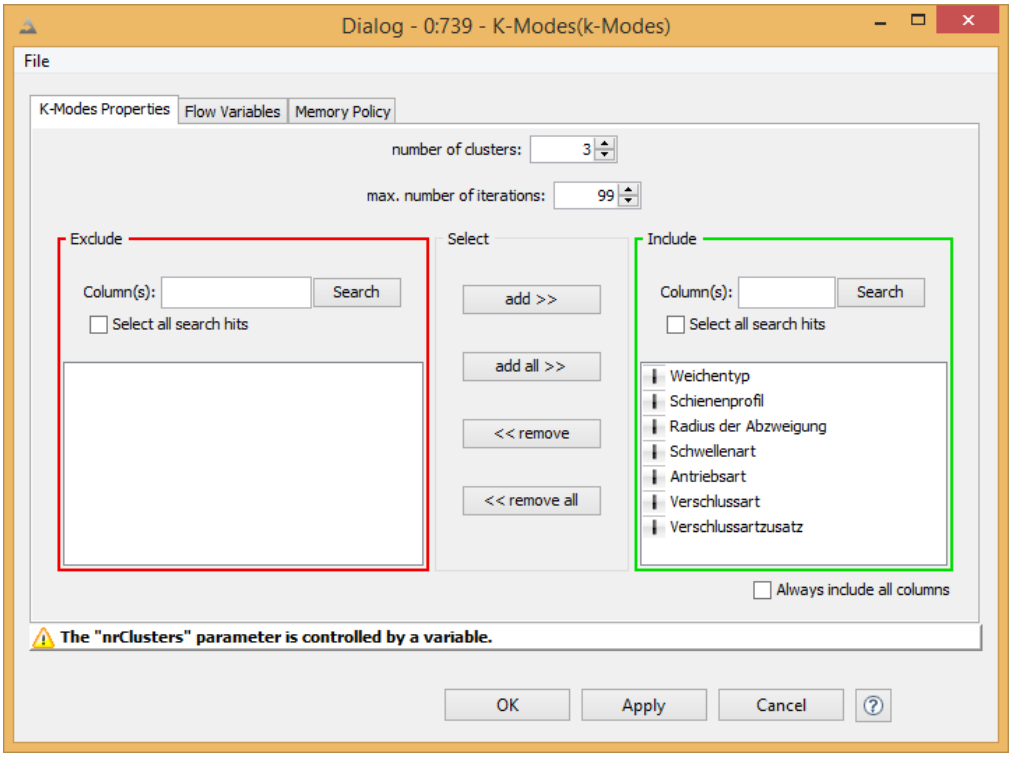

Abbildung 5.6.2: k-Modes Knoten - Konfiguration [KNIME]

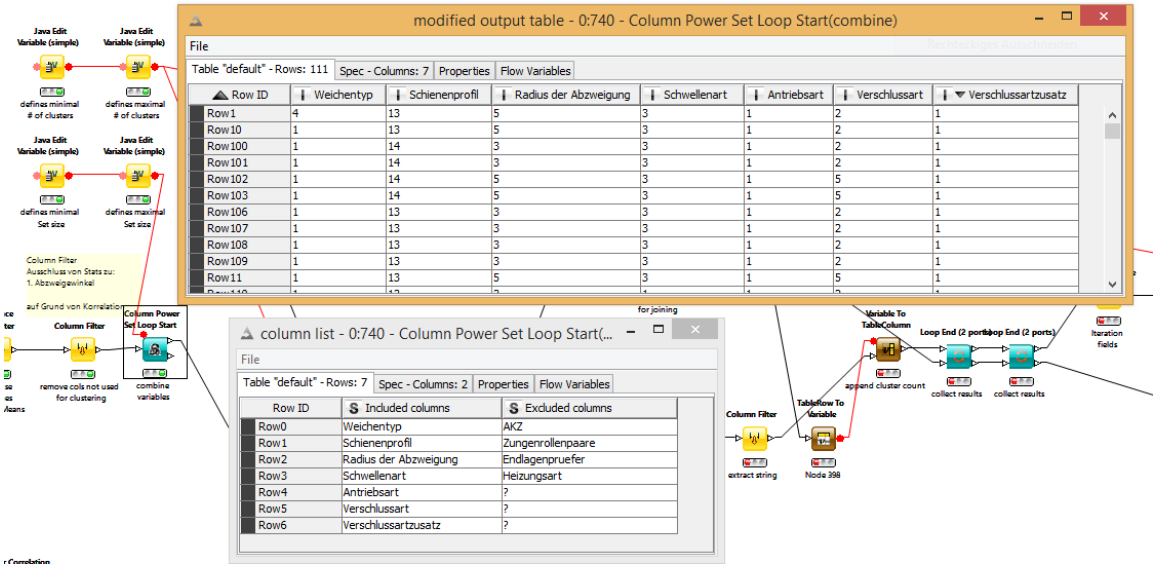

Abbildung 5.6.3: Knoten - Tupelbildung der Parameter für k-Modes [KNIME]

Kapitel 6

Graph Clustering

Zählen die beiden in den Kapiteln 5.4 und 5.5 vorgestellten Verfahren k-Modes und k-Means zu den bewährten konventionellen Partitionsmethoden der Clusteranalyse, so soll nun die moderne Disziplin des Graph Clustering vorgestellt werden.
Dem Graph Clustering bzw. den graphentheoretischen Clusteringverfahren liegt die Instanz als strukturiertes Netzwerk von Knoten, die untereinander durch Kanten verknüpft sind zu Grunde [26] (Abbildung 6.0.1). Diese Kanten werden in der Regel gewichtet und spiegeln so die Distanz bzw. Ähnlichkeit der beiden verbundenen Objekte wider. Häufig findet das Graph Clustering für Objektgruppen Anwendung, in denen nur ein Bruchteil der Objekte verbunden sind, was bei der Instanz dieser Arbeit nicht der Fall ist.

Der Graph wird unter Berücksichtigung der Größen (Kapitel 4) wie folgt definiert [2]:

$$g = (V, E, \omega) \quad \text{mit:} \tag{6.0.1}$$

$$(i) \quad V \text{ sei eine Menge von } M = 111 \text{ Knoten } v_j \tag{6.0.2}$$

$$(ii) \quad E \text{ sei eine Menge von } \frac{M(M-1)}{2} = 6105 \text{ Kanten } e_{ij} \text{ mit}$$

$$e_{ij} = \{v_i; v_j\} \qquad v_i, v_j \in V \tag{6.0.3}$$

$$(iii) \quad \omega \text{ sei eine Funktion der Kantengewichte und}$$

$$\omega(\{v_i; v_j\}) \in [0; 1] \tag{6.0.4}$$

Moderne Methoden des Graph Clustering unterscheiden sich in ihren Heuristiken von konventionellen Methoden. Bei der Suche nach der Optimallösung eines Optimierungsproblems kommen Metaheuristiken als Suchmethoden zum Einsatz, die universal für zahlreiche Problemstellungen verwendet werden können.

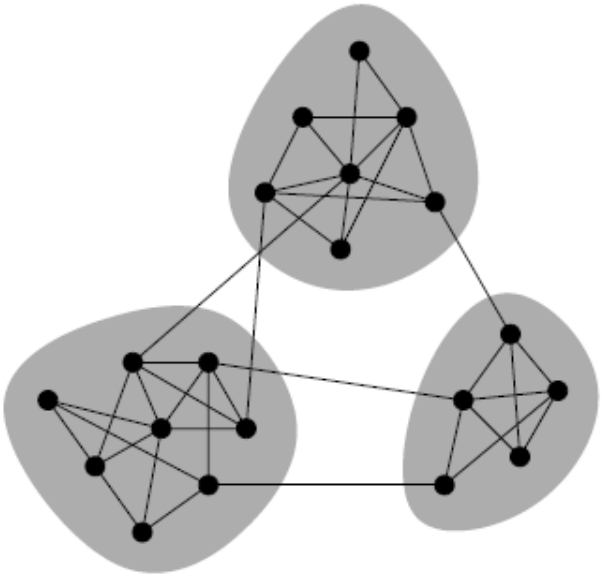

Abbildung 6.0.1: Beispielhaftes Netzwerk von (verbundenen) Objekten [19]

Der zehnte DIMACS Implementation Challenge Workshop mit dem Titel „Graph Partitioning and Graph Clustering“ veröffentlichte einige Ansätze und Methoden des Graph Clustering zur Lösung von Optimierungsproblemen [4]. Den Teilnehmern des Wettbewerbs wurden dabei Instanzen unterschiedlicher Größe und Struktur vorgelegt, welche unter Verwendung eines Solvers gelöst werden mussten. Eine Gegenüberstellung der Resultate brachte Aufschluss über die Performance der Solver, wobei zwei Disziplinen Pareto und Quality Challenge bewertet wurden.
Während in der Pareto Challenge die Algorithmen bei Erreichen einer maximalen Iterationszahl oder einer Stunde Rechenzeit abgebrochen wurden, stellte bei der Quality Challenge die Rechenzeit die einzige Stoppbedingung dar.
Die Metaheuristik Variable Neighborhood Search (VNS) lieferte in beiden Fällen herausragende Ergebnisse und eine beachtliche Effizienz, so dass sich diesem Verfahren im folgenden Kapitel intensiv gewidmet werden soll.

6.1 Variable Neighborhood Search

Variable Neighborhood Search ist eine *Metaheuristik* bzw. ein *Framework* für die Erstellung einer Heuristik, mit dem Ziel kombinatorische und globale Optimierungsprobleme zu lösen. Ihr Grundprinzip besteht aus der systematischen Änderung von Nachbarschaften kombiniert mit einer lokalen Suche [11].
„Neighborhood“ (deutsch: „Nachbarschaft“) N meint an dieser Stelle die Nach-

barschaft einer Lösung, wobei eine Lösung durch ihre Clusterzentroide und der Objektzugehörigkeiten bestimmt ist. Die Metaheuristik VNS basiert auf drei Fakten:

(1) Ein lokales Optimum gemäß einer Nachbarschaftsstruktur muss nicht ebenso lokal optimal sein für eine andere Nachbarschaftsstruktur.

(2) Ein globales Optimum ist ein lokales Optimum gemäß aller möglichen Nachbarschaftsstrukturen.

(3) Für viele Probleme gilt: Lokale Optima gemäß einer oder mehrere Nachbarschaftsstrukturen liegen oft relativ nahe beieinander.

Tabelle 6.1.1: Variable Neighborhood Search Fakten [2]

Eine Lösung y_L eines Optimierungsproblems gilt als lokales Optimum, wenn

$$f(y_L) > f(y) \qquad \forall y \in N(y_L) \tag{6.1.1}$$

mit $f(\cdot)$ sei eine Gütemaßfunktion für Clusterlösungen

Für die Definition einer Gütefunktion $f(\cdot)$ wird auf das folgende Kapitel (6.2) verwiesen.

Die Variabilität in Nachbarschaftsstrukturen wird durch Techniken der Tabelle 6.1.2 gewährleistet. Eine Bewertung dieser muss vor Durchführung des Verfahrens stattfinden, da die Einschränkungen der Clusterzahl k (5.2.2) zu berücksichtigen sind. Bei jeder der gelisteten Methode wird die Clusterzahl k variiert. Entsprechend wird die Metaheuristik VNS den hierarchischen Verfahren zugeordnet.
Die Techniken beschreiben sehr abstrakt Suchmethoden, um alternative Lösungen zu finden.

(1) *Singleton*: Ein Cluster wird aufgelöst und alle enthaltenen Knoten bilden jeweils ihr eigenes Cluster

(2) *Division*: Ein Cluster wird in zwei Gruppen zu gleichen Teilen geteilt. Die Knotenzuordnung erfolgt zufällig.

(3) *Neighbor*: Ein Cluster wird aufgelöst und alle Knoten werden einem benachbartem oder neuem Cluster zugeordnet.

(4) *Fusion*: Zwei oder mehr Cluster werden vereint.

(5) *Redistribution*: Ein Cluster wird aufgelöst und alle enthaltenen Knoten werden zufällig einem benachbartem Cluster zugeordnet.

Tabelle 6.1.2: Techniken der Nachbarschaftsveränderungen [2]

Die Methode der VNS zeichnet sich durch eine Verbesserungsheuristik und der Erkundung von Nachbarschaftsstrukturen für Clusterlösungen aus (Tabellen 6.1.1 und 6.1.2). Die durchsuchte Nachbarschaft weitet sich mit jedem Iterationsschritt aus, der zu keiner neuen optimalen Lösung führt.

Die Verbesserungsheuristik wird durch einen Algorithmus der Label Propagation [1] bewerkstelligt. Der Begriff „Label“ steht für *Cluster*, welches die zugewiesenen Objekte und den Clusterzentroiden referiert. „Propagation“ impliziert eine Aktualisierung und Ausweitung eines aktuellen Labels, indem alle enthaltenen Objekte probeweise anderen Labels zugewiesen werden. Steigert dies die Güte der Clusterlösung, werden die bestehenden Labels durch die neuen aktualisierten ersetzt. Bei diesem Verfahren wird im Gegensatz zur Suchmethodik der Nachbarschaftsveränderung explizit eine Verbesserung der Clusterlösung angestrebt.

Eine spezielle Variante der VNS stellt die Variable Neighborhood Decompostion Search dar. Sie erlaubt eine schnellere Erkundung des Suchraums, da nur ein Teil der Clusterlösung auf Verbesserung untersucht wird [2]. Der Pseudo-Algorithmus des Verfahrens wird im Anhang aufgeführt (A.1).

6.2 Maximierung der Modularität

Zunächst von Girvan und Newman [17] als Stoppregel für eine hierarchische divisive Heuristik eingeführt, wurde die Modularität später zu einem unabhängigen Kriterium, welches eine Bestimmung von optimalen Partitionen sowie einen Vergleich zwischen Partitionen erlaubt, die mit Hilfe verschiedener

Verfahren gewonnen wurden [2].
In dieser Arbeit dient das Kriterium der Modularität deshalb als Gütemaß für Ergebnisse bzw. Lösungen der Clusteranalyse. Die Berechnung des Maßes fand bereits in Kapitel 6.1 als mathematische Funktion $f(\cdot)$ Erwähnung.

Die optimale Partition einer Objektmenge V in einem Graphen $G = (V, E, \omega)$ maximiert über alle Cluster die Summe der Gewichte aller vollständig enthaltenen Kanten abzüglich der erwarteten Menge, sollten die Kanten zufällig angeordnet werden [2]. Die Summen der Kantengewichte müssen dabei relativ zu allen Gewichten berücksichtigt werden.

Sei $G = (V, E, \omega)$ ein Graph und C_V eine Partition der Menge V.

$$Q = \sum_{C \in C_V} e_C - \hat{e}_C \qquad \text{mit} \tag{6.2.1}$$

e_C ... Kantengewichte $\omega(\{u; v\})$ mit beiden Knoten $u, v \in V_C$ und
$\hat{e}_C$ ... erwartete Kantengewichte mit beiden Knotenpunkten in V_C.

$$Q = \frac{\sum_{C \in C_V} \sum_{\{u;v\} \in E_C} \omega(\{u; v\})}{\sum_{e \in E} \omega(e)} - \frac{\sum_{C \in C_V} \left(\sum_{v \in V_C} \omega(v) \right)^2}{4 \left(\sum_{e \in E} \omega(e) \right)^2} \quad \text{mit} \tag{6.2.2}$$

V_C ... Menge der Knoten v in C und
E_C ... Alle Kanten $e = \{u; v\} \in E$ mit $u, v \in V_C$ und

$$\omega(v) = \sum_{u \in V} \omega(\{u; v\}) \quad \forall v \in V. \tag{6.2.3}$$

Man definiere eine quadratische Matrix $\boldsymbol{e}$ der Dimension $k \times k$ (k sei die Clusterzahl) mit den $\mathbb{R}$-wertigen Einträgen e_{ij}, die den Anteil der Kantengewichte, welche zwischen den Clustern i und j ($i, j = 1, \ldots, k$) verlaufen als Wert haben [18].
Die Spur $Tr(\boldsymbol{e})$ beschreibt die Summe von k Einträgen auf der Diagonalen der Matrix, welche jeweils die Gewichte der Kanten mit beiden Knotenpunkten in einem Clustern als Summanden vorweisen.

$$Tr(\boldsymbol{e}) = \sum_{i=1}^{k} e_{ii} \tag{6.2.4}$$

Weiterhin seien die Zeilensummen der Matrix $\boldsymbol{e}$

$$a_i = \sum_{j=1}^{k} e_{ij} \qquad \forall i = 1, \ldots, k. \tag{6.2.5}$$

An dieser Stelle sei angemerkt, dass Gewichte derjenigen Kanten, welche eine Verbindung zweier Cluster i und j ($i \neq j$) darstellen, zu gleichen Teilen auf die Einträge e_{ij} und e_{ji} aufgeteilt werden müssen ($e_{ij} = e_{ji}$). Diese Eigenschaft führt zur Symmetrie der Matrix ($\boldsymbol{e}^T = \boldsymbol{e}$). Ebenso entspricht die Summe der Einträge in der i-ten Zeile a_i der i-ten Spaltensumme für alle $i = 1, \ldots, k$.

In einem Netzwerk, in dem Kanten zwischen Knoten zufällig, d.h. ohne Berücksichtigung der Cluster denen sie angehören gebildet werden, wären die Einträge wie folgt definiert [18]:

$$\hat{e}_{ij} = a_i \, a_j \qquad \forall i, j = 1, \ldots, k. \tag{6.2.6}$$

Die Modularität berechne sich entsprechend [18]

$$Q = \sum_{i=1}^{k} \left(e_{ii} - a_i^2 \right) = Tr(\boldsymbol{e}) - ||\boldsymbol{e}^2||, \tag{6.2.7}$$

wobei $||\boldsymbol{x}||$ die Summe der Elemente der Matrix $\boldsymbol{x}$ ist.

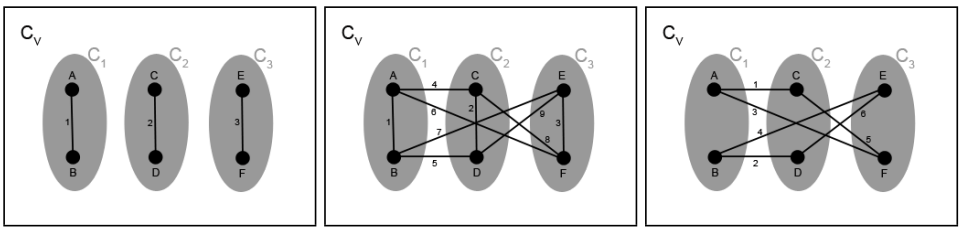

Abbildung 6.2.1: Beispiele für Graphen partitionierter Netzwerke

Die drei Graphen der Abbildung 6.2.1 illustrieren jeweils eine Partition eines Netzwerks mit 6 Knoten $(A, \ldots, F)$ und $k = 3$ Clustern. Die Kantengewichte seien konstant $\omega(e) = 1$ für alle $e \in E$.
Beispiel 1 führt zu einer maximalen Modularität und damit optimalen Partition, da alle Knoten eines Clusters untereinander verbunden sind und keine clusterübergreifende Verknüpfungen existieren. Für die Modularität ergibt sich ein Wert von$< Q = \frac{2}{3}$.

Beispiel 2 zeigt eine zufällige Anordnung der Kanten, indem alle Knoten mit jeweils einem Knoten eines jeden Clusters verbunden sind. Die Modularität ist in diesem Fall $Q = 0$.
Beispiel 3 liefert die Modularität $Q = -\frac{1}{3}$. Innerhalb den 3 Clustern existieren keine Verbindungen, jedoch außerhalb zu den jeweils anderen beiden Clustern.

Die Modularität Q weist einen Wertebereich von $[-\frac{1}{2}; \frac{k-1}{k}]$ auf. Zur Verwendung als Gütemaß für Partitionen von Objektmengen innerhalb eines Netzwerks scheint eine Normierung auf den Wertebereich $[0; 1]$ sinnvoll. Ebenso ist diese Anpassung notwendig, um Ergebnisse verschiedener Clusterzahlen vergleichen zu können.

$$Q^* = \begin{cases} Q \cdot \frac{k}{k-1} & \text{für } Q > 0 \\ 0 & \text{für } Q \leq 0 \end{cases} \tag{6.2.8}$$

$$Q^* \in [0; 1] \qquad \forall k \in \mathbb{N}, \text{ besonders } k \in [3; 10] \tag{6.2.9}$$

Es fällt auf, dass der Wert $Q^* = 0$ alle negativen Werte von Q bündelt. Ein Vergleich innerhalb dieses Wertebereichs mit Hilfe von Q^* ist daher nicht möglich. Diese Problematik kann getrost vernachlässigt werden, da das Optimierungsproblem *Maximierung der Modularität* lautet und ein Wert größer 0, nahe 1 angestrebt wird.

Die in Kapitel 5.5 definierte Zielfunktion des k-Means Clusteringverfahrens unterscheidet durch die fehlende Berücksichtigung von clusterübergreifenden Objektverbindungen von dem Gütemaß der Modularität. Während bei k-Means das Ziel „Minimierung der Distanzen zwischen Zentroid und Objekte innerhalb der Cluster“ lautet, erwartet man bei einer „Modularität-besten“ Lösung, dass die Distanzen zwischen Objekten innerhalb eines Clusters erheblich niedriger ausfallen als die Distanzen zwischen zwei Clustern.

In der vorliegenden Instanz sind alle Knoten paarweise miteinander verbunden. Entsprechend weist der Graph $M = 111$ Knoten und 6105 Knotenpaare bzw. gewichtete Kanten auf (6.0.2 und 6.0.3). Alle Einträge der Matrix $\boldsymbol{e}$ sind größer 0, ausgenommen dem Fall, dass Cluster i lediglich einen Knoten beinhaltet, was zum Eintrag $e_{ii} = 0$ führen würde.

Die besten Ergebnisse des Clusterings der Messwerte (Kapitel 5.5) sollen nun herangezogen werden, um zu prüfen ob durch eine angepasste VNS Metaheuristik eine Verbesserung der Partitionen erreicht werden kann.
Die Durchführung und Auswertung des Graph Clustering soll im folgenden Kapitel vor einem technisch-operativem Hintergrund erläutert werden, da die Adaption der abstrakten Methodiken der VNS für das verwendete Datenanalysetool eine besondere Herausforderung darstellte.

6.3 Umsetzung in KNIME

Die Intention der Anwendung einer Graph Clusteranalyse auf die Instanz dieser Arbeit bestand zum Einen im Kontrast zwischen den konventionellen Ansätzen (Kapitel 5) und dem moderneren Verfahren des Graph Clustering. Zum Anderen ist von Interesse, wie performant Methoden des Graph Clustering mit Instanzen operieren, die sich durch eine paarweise Verknüpfung sämtlicher Objekte auszeichnen. Schließlich ist diese Methode für Netze mit wenigen Kanten prädestiniert.

Als Input dienten die besten k-Means Lösungen, wobei diese zunächst entsprechend der Modularität bewertet und der Wert anschließend gemeinsam mit der Objektstruktur an den Initialisierungsknoten der VNS Route übergeben wurden.

Die Optionen und Techniken der Nachbarschaftsveränderung (Tabelle 6.1.2) lassen per Definition eine große Variation der Ausgangslösung zu. Dieser Umstand steht im Gegensatz zu den aus Kapitel 5 bekannten Methoden, welche meist in einem zur Initiallösung benachbarten lokalen Optima resultieren. Daher war es erforderlich, den Techniken der Nachbarschaftsveränderung einen größtmöglichen Spielraum zu lassen ohne dabei die (Neben-)Bedingungen (5.5.2) außer Acht zu lassen.

Ebenso sollte darauf geachtet werden, dass die Clusterzahl bei der Suche nach neuen Lösung beibehalten wird. Dies wurde im Workflow durch eine Abfolge von zwei Techniken gewährleistet, wobei zunächst die Clusterzahl k um 1 verringert und anschließend um 1 erhöht wurde (Abbildung 6.3.1).
Die Wahl der Techniken fiel dabei unter Berücksichtigung der problemlosen Implementierung und des Rechenaufwands auf die Verfahren der *Fusion* oder *Redistribution* im ersten Fall sowie der *Division* für die Erhöhung der Clusterzahl.

Die generierte neue Lösung wurde im nächsten Schritt mittels der Label Propagation Algorithmus (LPA) analysiert. An dieser Stelle war ein Loop-Kreislauf notwendig, um jedes Cluster einzeln auf eine bessere Objektstruktur untersuchen zu können (Abbildung 6.3.2). Diese Überprüfung wurde für jedes Objekt durchgeführt, indem nach einem Cluster gesucht wurde, welches dem Objekt näher ist, d.h. zu dem eine stärkere Bindung besteht. Die Auswertung der per VNS generierten Lösungen ergab, dass in jedem Fall im Zuge der Verbesserungsheuristik LPA mindestens ein Objekt versetzt wurde.

Zum Abschluss eines Iterationsschrittes wird die Modularität der neuen Lösung berechnet (Abbildung 6.3.2), um die Qualität zu beziffern. Übersteigt der Wert dieser Lösung den der aktuellen Inputlösung, so wird letztere durch die

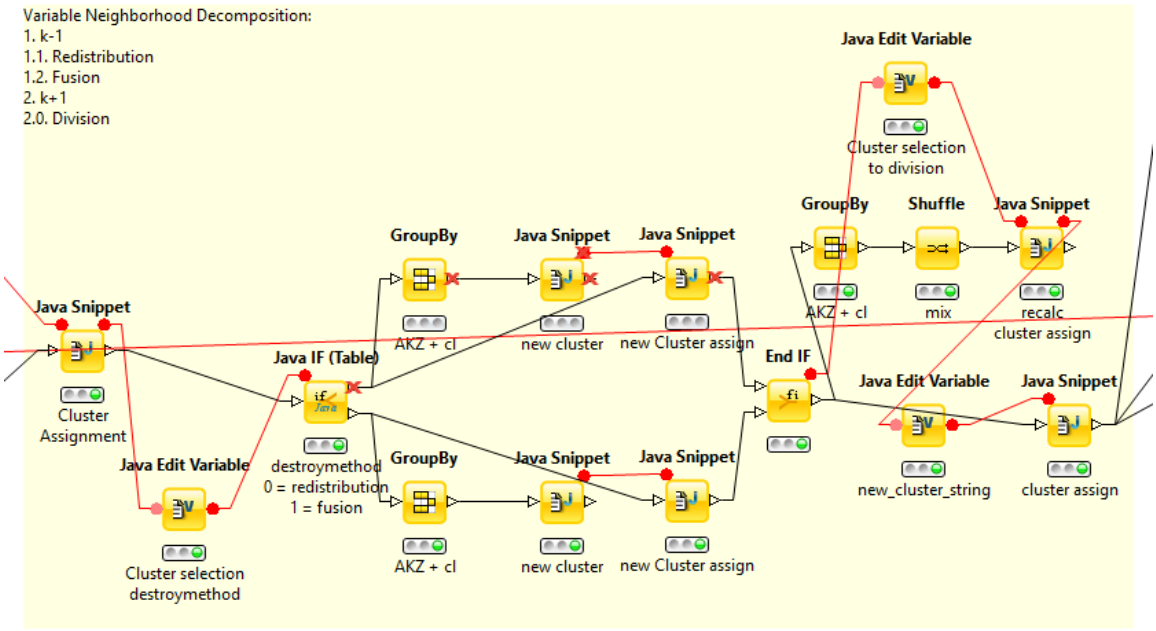

Abbildung 6.3.1: Routen der Nachbarschaftsveränderung [KNIME]

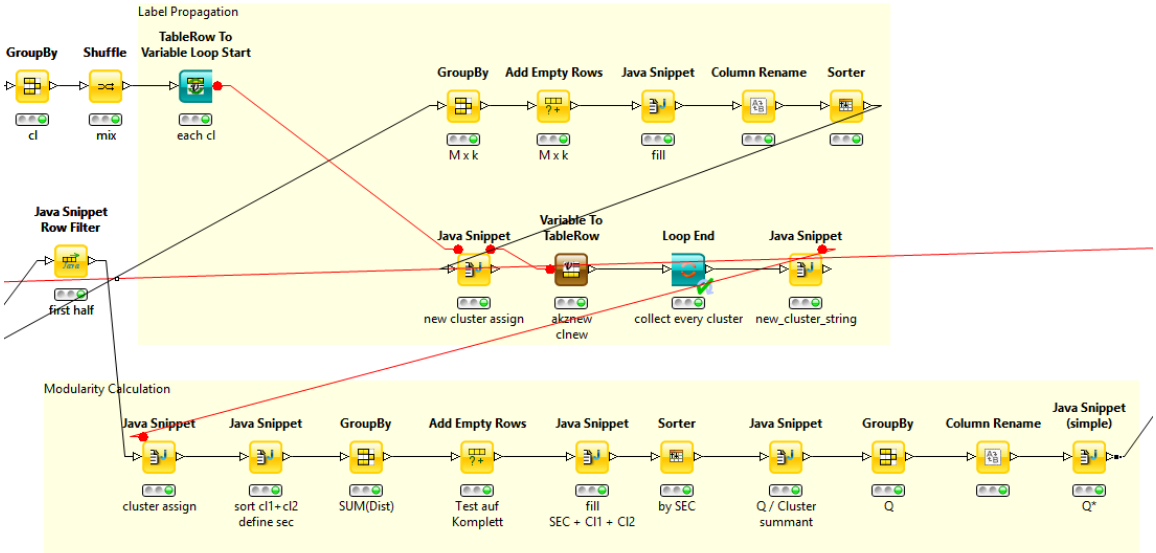

Abbildung 6.3.2: Routen der Label Propagation und Modularitätberechnung

neue ersetzt und an den Initialisierungsknoten übergeben. Stellt die neue Lösung hingegen keine Verbesserung dar, so wird erneut mit der „alten“ Lösung im folgenden Iterationsschritt weiter verfahren.

In KNIME wird die optionale Änderung des Inputs eines Loops durch einen Knoten namens „Delegating Loop End“ ermöglicht. Der Abschlussknoten des Kreislaufs weist zwei Input-Schnittstellen auf, wobei eine mit der Ausgangstabelle für die folgende Iteration verknüpft werden muss und die zweite der Datensammlung dient.
Die maximale Zahl an Iterationen, die im Zuge der Suche nach besseren Lösungen durchgeführt werden wird an den Abschlussknoten übermittelt und bei Erreichen dieser Zahl werden die generierten Lösungen gespeichert und der Kreislauf wird mit der nächsten Ausgangslösung als erste Inputtabelle fortgeführt. Alternativ wird der Kreislauf im Falle von mehreren aufeinanderfolgenden Iterationen ohne Verbesserung der Lösung abgebrochen.

Die Ergebnisse der Tabelle 6.3.1 liefern Aufschluss über die Iterationszahl und die Art der in Kraft getretenen Stoppbedingung.
Im Mittelpunkt der Auswertung soll jedoch die relative bzw. absolute Verbesserung der Ausgangslösung stehen. Die Auswertung der Resultate ergab, dass in keinem Fall eine exorbitante Steigerung der Modularität eintritt. Dies kann zum Einen darauf zurückgeführt werden, dass die Methoden der Nachbarschaftsveränderung Restriktionen unterliegen und zum Anderen ist die Instanz durch die vollständige Verknüpfung der Objekte untereinander bedingt für die VNS Methode geeignet.
So bleibt zu konstatieren, dass das Verbesserungspotenzial der Lösungen eher gering ausfällt. Auf der anderen Seite muss festgehalten werden, dass das Verfahren lediglich bei 14 (0,175%) Lösungen zu keinerlei Verbesserungen geführt hat.

		Clusterzahl k							
		3	4	5	6	7	8	9	10
Minimum	korrigierte Modularität	0,0351	0,0256	0,0202	0,0165	0,0121	0,0102	0,0083	0,0074
Mittelwert		0,0387	0,0275	0,0239	0,0196	0,0155	0,0131	0,0105	0,0089
Maximum		0,0405	0,0318	0,0296	0,0222	0,0213	0,016	0,014	0,0115
Minimum	absolute Verbesserung	0	0	0,0011	0	0	0	0	0
Mittelwert		0,0035	0,001	0,003	0,0024	0,0029	0,0015	0,0016	0,0011
Maximum		0,0051	0,0046	0,0066	0,0053	0,0087	0,0038	0,0047	0,0041
Minimum	relative Verbesserung	0	0	0,0622	0	0,0071	0	0	0
Mittelwert		0,1588	0,0429	0,1982	0,1745	0,2879	0,1559	0,218	0,1612
Maximum		0,2372	0,2415	0,4581	0,3965	0,9056	0,3961	0,6113	0,6702
Minimum	Iterationszahl	20	20	22	20	21	20	20	20
Mittelwert		36,1	22,9	34,7	29,3	27,2	26,4	30,6	29,1
Maximum		61	47	47	49	47	40	54	68

Tabelle 6.3.1: Resultate des Graph Clustering

Kapitel 7

Clusterzusammenführung

Im Anschluss an die Clusteranalyse des Kapitels 5, durchgeführt mit den Verfahren k-Means für Messwerte respektive k-Modes für Weichenstammdaten, sollen die Partitionen im Folgenden verglichen werden. Es stellt sich die Frage, in wie weit Objekte, die auf der einen Seite als „ähnlich“ eingestuft wurden auch auf der anderen Seite „benachbart“ sind.

Dementsprechend ist es bei der Zusammenführung der Partitionen wünschenswert, dass die Verfahren k-Means und k-Modes ähnliche Objektstrukturen aufweisen. Um eine Aussage darüber treffen zu können wird eine Häufigkeitsmatrix U eingeführt (7.0.1 beispielhaft), deren Elemente die Anzahl der Objektübereinstimmungen in Messdaten- und Weichenstammdaten-Clustern beschreibt.
Als Gütemaß einer Clusterzusammenführung wird anschließend der Kontingenzkoeffizient nach Pearson eingeführt und ausführlich erläutert.

$$U = \begin{array}{c} \\ \text{Means}_1 \\ \text{Means}_2 \\ \text{Means}_3 \\ \text{Means}_4 \end{array} \begin{array}{c} \begin{array}{cccc} \text{Modes}_1 & \text{Modes}_2 & \text{Modes}_3 & \text{Modes}_4 \end{array} \\ \begin{pmatrix} 5 & 25 & 2 & 4 \\ 8 & 6 & 10 & 3 \\ 0 & 1 & 2 & 0 \\ 32 & 5 & 5 & 3 \end{pmatrix} \end{array} \tag{7.0.1}$$

$$k = 3, \quad U \in \mathbb{N}^{4\times 4}$$

7.1 Häufigkeitsmatrix und Kontingenzkoeffizient

Bei der Auswahl geeigneter Clusteranalyseverfahren wurde die Klasse der Partitionsmethoden in den Fokus gerückt. Hintergrund für diese Restriktion ist

die Kontrolle der Clusterzahl (Kapitel 5.2). k wird dabei wie folgt vorgegeben (5.2.2):

$$k \in [3;10], \quad k \in \mathbb{N}$$

Mit Zunahme von k wächst das Maß an Differenzierung in der Gesamtheit der Variablenwerte. Beim Arbeiten mit kategorischen Variablen, deren Domains typischer Weise eine geringe Mächtigkeit aufweisen (5.2.1) wirkt sich eine Änderung der Größe k direkter aus als auf die kontinuierlichen Wertebereichen der numerischen Variablenwerte, welche mehr Möglichkeiten der Differenzierung und ein „feineres“ Distanzmaß bieten.

Der Clusterzusammenführung liegt die Suche nach Übereinstimmungen von Messwert- und Stammdaten-spezifischen Eigenschaften zu Grunde. Objektpaare bzw. Objekttupel, denen in zahlreichen Clusterergebnissen beider Verfahren eine Ähnlichkeit zugeschrieben wird müssen abgeglichen werden. Dies geschieht durch Gegenüberstellung ihrer jeweiligen Eigenschaften und ihrem technischen Verhalten. Hierfür werden die Variablenwerte der Clusterzentroide, welche die entsprechenden Objektgruppen beinhalten verglichen und direkt in Bezug zueinander gesetzt.
Die Entscheidung, dass k_r als Clusterzahl eines k-Means Ergebnisses und k_s, dem Pendant auf Seite der Clusteranalyse nach Weichenstammdaten übereinstimmen müssen, beruht auf der Idee einer vergleichbaren Granularität in der Clusterdifferenzierung.
Illustriert wird diese Einschränkung graphisch, indem die Anzahl der Clusterzentroide und die mittlere Objektzahl pro Cluster gleich groß sein müssen (Abbildung 7.1.1).

Die Häufigkeitsmatrix $\boldsymbol{U}^{k\times k}$ sei eine $\mathbb{N}$-wertige Matrix mit Einträgen u_{k_r,k_s}, für die gilt:

$$(i) \quad u_{k_r,k_s} \in \mathbb{N}_0 \text{ mit } k_r, k_s \in [1,k],\ 0 \leq u_{k_r,k_s} \leq M-k+1 \tag{7.1.1}$$

$$(ii) \quad u_{k_r,k_s} = \sum_{m=1}^{M} r_{m,k_r} \cdot s_{m,k_s}, \tag{7.1.2}$$

$$\upsilon_{k_r,k_s} = \{m : (r_{m,k_r} = 1) \cap (s_{m,k_s} = 1) : m = 1,...,M\}, \tag{7.1.3}$$

$$u_{k_r,k_s} = |\upsilon_{k_r,k_s}| \qquad \forall k_r = 1,\ldots,k\,;\, k_s = \{1,\ldots,k\} \tag{7.1.4}$$

$$(iii) \quad \sum_{k_r=1}^{k}\sum_{k_s=1}^{k} u_{k_r,k_s} = M \tag{7.1.5}$$

Die Zugehörigkeitsindikatoren r_{m,k_r} und s_{m,k_s} wurden in den Kapiteln 5.4 bzw. 5.5 eingeführt (5.4.12 und 5.5.12).

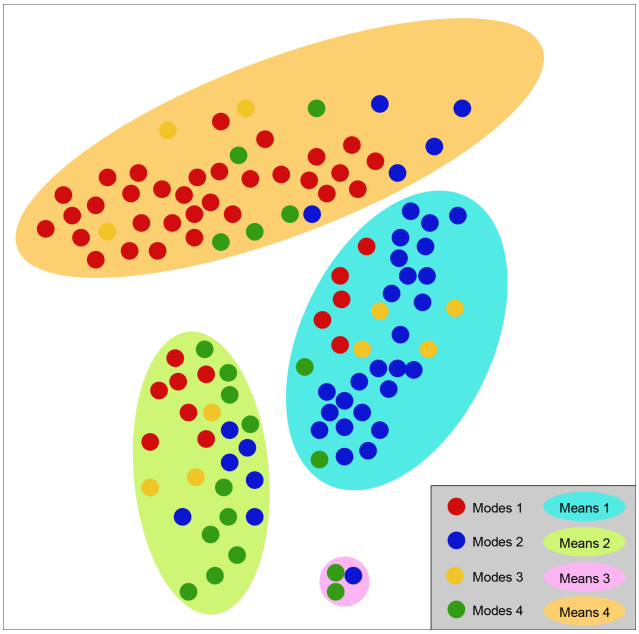

Abbildung 7.1.1: Netzwerk der Clusterzusammenführung

Man definiere zusätzlich die Zeilen- und Spaltensummen sowie die erwartete Häufigkeit bzw. Wahrscheinlichkeit:

$$(i) \quad \text{Spaltensumme: } u_{*,k_s} = \sum_{k_r=1}^{k} u_{k_r,k_s} \quad \forall\ k_s = 1,\dots,k \tag{7.1.6}$$

$$(ii) \quad \text{Zeilensumme: } u_{k_r,*} = \sum_{k_s=1}^{k} u_{k_r,k_s} \quad \forall\ k_r = 1,\dots,k \tag{7.1.7}$$

$$(iii) \quad \text{Erwartete Häufigkeit: } \hat{u}_{k_r,k_s} = \frac{u_{*,k_s} \cdot u_{k_r,*}}{M} \tag{7.1.8}$$

(iv) Erwartete Wahrscheinlichkeit für die Zugehörigkeit eines Objekts W_m:

$$p_{k_r,k_s} = \frac{u_{*,k_s} \cdot u_{k_r,*}}{M^2} \tag{7.1.9}$$

An dieser Stelle sollen zur Vereinfachung weiterführender Berechnungen mit der Häufigkeitsmatrix U die Indizes k_r und k_s zusammengeführt werden. Dabei wird die Matrix- in eine Vektorform überführt, indem die Zeilenvektoren in aufsteigender Reihenfolge aneinander gereiht werden:

$$\begin{aligned} &j = (k_r - 1)\,k + k_s \text{ und es gilt } \hat{k} = k^2, \\ &\text{so dass } j \in [1;\hat{k}] \end{aligned} \tag{7.1.10}$$

Für die bisher in diesem Kapitel eingeführten Größen gelte der Substitution 7.1.10 entsprechend:

$$
\begin{aligned}
u_j &= u_{k_r,k_s}, \\
u_{k_{r*}} &= u_{k_r,*}, \\
u_{k_{*s}} &= u_{*,k_s}, \\
v_j &= v_{k_r,k_s}, \\
p_j &= p_{k_r,k_s}, \\
\hat{u}_j &= \hat{u}_{k_r,k_s} \qquad \forall k_r, k_s = 1, \ldots, k
\end{aligned}
$$

Mit dem Ziel im Zuge einer Zusammenführung zweier Clusterergebnisse ähnliche Objektstrukturen zu erhalten, geht die Notwendigkeit eines Maßes für die Übereinstimmung der Partitionen einher. Der Kontingenzkoeffizient nach Karl Pearson [21] erfüllt diese Anforderung und analysiert die Einträge der Häufigkeitsmatrix. Er bestimmt die Abweichung der realen Häufigkeitswerte von den erwarteten Werten.
Dem Kontingenzkoeffizient liegt die Frage nach der Unabhängigkeit der verknüpften Resultate zu Grunde. Von Unabhängigkeit wird dann gesprochen, wenn die realen Werte den erwarteten Werten sehr ähnlich sind. Hingegen weist ein hoher Wert auf eine ausgeprägte Abhängigkeit der Einordnungen hin. Im Kontext dieser Arbeit wird die Abhängigkeit der beiden Partitionierungen und demzufolge ein hoher Wert angestrebt.

1. Allgemeiner Kontingenzkoeffizient [3]:

$$\chi^2 = \sum_{j=1}^{\hat{k}} \frac{(u_j - \hat{u}_j)^2}{\hat{u}_j}, \tag{7.1.11}$$

wobei gilt:

$$\chi^2 \in [0, M \cdot k - M] \tag{7.1.12}$$

2. Kontingenzkoeffizient nach Pearson:

$$C = \sqrt{\frac{\chi^2}{\chi^2 + M}}, \quad \text{mit} \quad C \in \left[0, \sqrt{\frac{k-1}{k}}\right] \tag{7.1.13}$$

3. Korrigierter Kontingenzkoeffizient:

$$C_{korr} = \sqrt{\frac{k}{k-1}} \cdot C, \quad \text{mit} \quad C_{korr} \in [0,1] \tag{7.1.14}$$

4. Zugehörigkeitsindikator:

$$I(m \in \upsilon_j) \in \{0,1\} \quad \forall m = 1, \ldots, M \tag{7.1.15}$$

und es gilt:

$$u_j = \sum_{m=1}^{M} I(m \in \upsilon_j) \tag{7.1.16}$$

Mit Einführung des Erwartungswertes erhält man

$$\mathbb{E}[I(m \in \upsilon_j)] = p_j \quad \forall m = 1, \ldots, M. \tag{7.1.17}$$

Und damit:

$$\mathbb{E}[u_j] = M p_j = \hat{u}_j, \text{ sowie} \tag{7.1.18}$$

$$\text{Var}(I(m \in \upsilon_j)) = p_j(1-p_j) \quad \forall m = 1, \ldots, M. \tag{7.1.19}$$

Der zentrale Grenzwertsatz [22] liefert

$$\frac{u_j - M p_j}{\sqrt{M p_j (1-p_j)}} = \frac{\sum_{m=1}^{M} I(m \in \upsilon_j) - M p_j}{\sqrt{M p_j (1-p_j)}} \rightarrow \mathcal{N}(0,1). \tag{7.1.20}$$

Durch Umformung erhält man

$$Z_j := \frac{u_j - M p_j}{\sqrt{M p_j}} = \frac{u_j - \hat{u}_j}{\sqrt{\hat{u}_j}} \rightarrow \sqrt{1-p_j}\, \mathcal{N}(0,1) = \mathcal{N}(0, 1-p_j). \tag{7.1.21}$$

Um die Verteilung des Terms (7.1.21) der an dieser Stelle in eine Zufallsvariable Z_j überführt wird zu beweisen [16], betrachtet man zunächst die Kovarianz zweier Zufallsvariablen Z_j und $Z_{j'}$, $j \neq j'$, $j, j' \in [1, \hat{k}]$.

$$\begin{aligned}
\mathrm{Cov}\left(Z_j, Z_{j'}\right) &= \mathbb{E}\left[\frac{u_j - M\,p_j}{\sqrt{M\,p_j}}\,\frac{u_{j'} - M\,p_{j'}}{\sqrt{M\,p_{j'}}}\right] \\
&= \frac{1}{M\,\sqrt{p_j\,p_{j'}}}\,(\,\mathbb{E}\left[u_j\,u_{j'}\right] - \mathbb{E}\left[u_j\,M\,p_{j'}\right] - \\
&\quad \mathbb{E}\left[u_{j'}\,M\,p_j\right] + M^2\,p_j\,p_{j'}) \\
&= \frac{1}{M\,\sqrt{p_j\,p_{j'}}}\,(\,\mathbb{E}\left[u_j\,u_{j'}\right] - M\,p_{j'}\,\mathbb{E}\left[u_j\right] - \\
&\quad M\,p_j\,\mathbb{E}\left[u_{j'}\right] + M^2\,p_j\,p_{j'}) \\
&= \frac{1}{M\,\sqrt{p_j\,p_{j'}}}\,(\,\mathbb{E}\left[u_j\,u_{j'}\right] - M\,p_{j'}\,M\,p_j - \\
&\quad M\,p_j\,M\,p_{j'} + M^2\,p_j\,p_{j'}\,) \\
&= \frac{1}{M\,\sqrt{p_j\,p_{j'}}}\,(\,\mathbb{E}\left[u_j\,u_{j'}\right] - M^2\,p_j\,p_{j'}\,)
\end{aligned}$$

An dieser Stelle soll detailliert auf den Ausdruck $\mathbb{E}[u_j\,u_{j'}]$ eingegangen werden, wobei bekannt ist, dass ein Objekt W_m nicht zwei verschiedenen Clusterkombinationen zugehören kann:

$$I(m \in \upsilon_j)\,I(m \in \upsilon_{j'}) \;=\; 0 \quad \begin{array}{l} \forall m = 1, \ldots, M, \\ j, j' = 1, \ldots, \hat{k}, \\ j \neq j' \end{array} \tag{7.1.22}$$

Mit der Unabhängigkeit der (Häufigkeits-)Cluster erhält man

$$\begin{aligned}
\mathbb{E}\left[u_j\, u_{j'}\right] &= \mathbb{E}\left[\left(\sum_{m=1}^{M} I(m \in \upsilon_j)\right)\left(\sum_{m'=1}^{M} I(m' \in \upsilon_{j'})\right)\right] \\
&= \mathbb{E}\left[\sum_{m=1}^{M}\sum_{m'=1}^{M} I(m \in \upsilon_j)\, I(m' \in \upsilon_{j'})\right] \\
&= \mathbb{E}\left[\sum_{m=1}^{M} I(m \in \upsilon_j)\, I(m \in \upsilon_{j'})\right] + \\
&\quad \mathbb{E}\left[\sum_{m=1}^{M}\sum_{\substack{m'=1 \\ m' \neq m}}^{M} I(m \in \upsilon_j)\, I(m' \in \upsilon_{j'})\right] \\
&= \mathbb{E}\left[\sum_{m=1}^{M}\sum_{\substack{m'=1 \\ m' \neq m}}^{M} I(m \in \upsilon_j)\, I(m' \in \upsilon_{j'})\right] \\
&= \sum_{m=1}^{M}\sum_{\substack{m=1 \\ l' \neq l}}^{M} \mathbb{E}\left[I(m \in \upsilon_j)\, I(m' \in \upsilon_{j'})\right] \\
&= M\,(M-1)\,\mathbb{E}\left[I(m \in \upsilon_j)\, I(m' \in \upsilon_{j'})\right] \\
&= M\,(M-1)\,\mathbb{E}\left[I(m \in \upsilon_j\right]\,\mathbb{E}\left[I(m' \in \upsilon_{j'})\right] \\
&= M\,(M-1)\,p_j\,p_{j'}
\end{aligned}$$

Eingesetzt in die Kovarianz, bleibt

$$\begin{aligned}
\mathrm{Cov}\left(Z_j, Z_{j'}\right) &= \frac{1}{M\,\sqrt{p_j\,p_{j'}}}\left(M\,(M-1)\,p_j\,p_{j'} \;-\; M^2\,p_j\,p_{j'}\right) \\
&= -\sqrt{p_j\,p_{j'}}
\end{aligned}$$

Weiterführend wird sich dem Kontingenzkoeffizienten χ^2 gewidmet, wobei die Verteilung dieser Größe von Interesse ist.

Es soll gezeigt werden, dass

$$\chi^2 = \sum_{j=1}^{\hat{k}} \frac{(u_j - M\,p_j)^2}{M\,p_j} = \sum_{j=1}^{\hat{k}} Z_j^2 \quad \sim \quad \chi^2_{\hat{k}-1} \quad . \tag{7.1.23}$$

Bekannt ist:

$$\mathbb{E}\left[Z_j^2\right] = \mathrm{Var}\left(Z_j\right) = 1 - p_j \quad \text{und} \tag{7.1.24}$$

$$\mathrm{Cov}\left(Z_j\,,\, Z_{j'}\right) = -\sqrt{p_j p_{j'}} \quad . \tag{7.1.25}$$

Bei der weiteren Beweisführung sollen geometrische Aspekte berücksichtigt werden.

Seien $\overrightarrow{g} = (g_j, \ldots, g_{\hat{k}})$ und $\overrightarrow{p} = (\sqrt{p_j}, \ldots, \sqrt{p_{\hat{k}}})$ zwei Vektoren. Es gilt

$$(i) \quad g_j, \ldots, g_{\hat{k}} \sim \text{u.i.v. } \mathcal{N}(0,1) \tag{7.1.26}$$

$$(ii) \quad \sum_{j=1}^{\hat{k}} p_j = |\overrightarrow{p}| = 1 \tag{7.1.27}$$

Im Folgenden soll der Vektor

$$\overrightarrow{g} - (\overrightarrow{g} \circ \overrightarrow{p})\,\overrightarrow{p} \tag{7.1.28}$$

betrachtet und der Beweis, dass dieser die gleiche gemeinsame Verteilung wie $(Z_1, \ldots, Z_{\hat{k}})$ hat geführt werden.

Man betrachte zwei Koordinaten des Vektors (7.1.28):

$$\textit{s-te:} \quad g_s - \sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_s} \tag{7.1.29}$$

$$\textit{t-te:} \quad g_t - \sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_t} \tag{7.1.30}$$

und berechne die Kovarianz:

$$
\begin{aligned}
&\mathbb{E}\left[\left(g_s - \sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_s}\right)\left(g_t - \sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_t}\right)\right] \\
&= \mathbb{E}[g_s\, g_t] - \mathbb{E}\left[g_s \left(\sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_t}\right)\right] - \mathbb{E}\left[g_t \left(\sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_s}\right)\right] + \\
&\mathbb{E}\left[\left(\sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_s}\right)\left(\sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_t}\right)\right] \\
&= -\mathbb{E}\left[g_s^2 \sqrt{p_s}\,\sqrt{p_t}\right] - \mathbb{E}\left[g_t^2 \sqrt{p_t}\,\sqrt{p_s}\right] + \sqrt{p_s}\,\sqrt{p_t}\;\mathbb{E}\left[\left(\sum_{j=1}^{\hat{k}} g_j^2\, p_j\right)\right] \\
&= -2\sqrt{p_s}\,\sqrt{p_t} + \sqrt{p_s}\,\sqrt{p_t} \sum_{j=1}^{\hat{k}} \mathbb{E}\left[g_j^2\right] p_j \\
&= -2\sqrt{p_s}\,\sqrt{p_t} + \sqrt{p_s}\,\sqrt{p_t} \\
&= -\sqrt{p_s}\,\sqrt{p_g} \\
&= -\sqrt{p_s\, p_t}
\end{aligned}
$$

Des Weiteren ermittle man den folgenden Erwartungswert:

$$
\begin{aligned}
\mathbb{E}\left[\left(g_s - \sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_s}\right)^2\right] &= \mathbb{E}\left[(g_s)^2\right] - 2\,\mathbb{E}\left[g_s \left(\sum_{j=1}^{\hat{k}} g_j \sqrt{p_j}\,\sqrt{p_s}\right)\right] + \\
&\quad p_s\,\mathbb{E}\left[\left(\sum_{j=1}^{\hat{k}} g_j^2\, p_j\right)\right] \\
&= 1 - 2\,\mathbb{E}\left[g_s^2 \sqrt{p_s}\,\sqrt{p_s}\right] + \\
&\quad p_s \left(\sum_{j=1}^{\hat{k}} \mathbb{E}\left[g_j^2\right] p_j\right) \\
&= 1 - 2p_s + p_s \\
&= 1 - p_s
\end{aligned}
$$

An dieser Stelle sollen die Erkenntnisse auf die Ausführungen zum Kontingenzkoeffizienten angewendet werden. Man kann folgende Konvergenz fest-

stellen:

$$\sum_{j=1}^{\hat{k}} \left(\frac{u_j - M\, p_j}{\sqrt{M\, p_j}} \right)^2 \quad \rightarrow \quad \sum_{i=1}^{\hat{k}} (\textit{s-te Koordinate})^2 \tag{7.1.31}$$

Wissend dass $\overrightarrow{p}$ ein Einheitsvektor ist, d.h.

$$|\overrightarrow{p}| = \sum_{j=1}^{\hat{k}} (\sqrt{p_j})^2 = 1 \tag{7.1.32}$$

will man den Vektor

$$\overrightarrow{V_2} = \overrightarrow{g} - (\overrightarrow{g} \circ \overrightarrow{p})\, \overrightarrow{p} \tag{7.1.33}$$

geometrisch interpretieren (Abbildung 7.1.2).

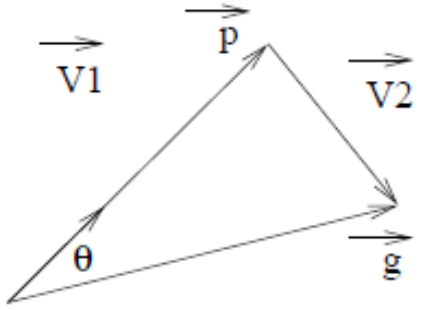

Abbildung 7.1.2: Projektion des Vektors g [16]

Hierfür definiere man den zweiten Summanden als

$$\overrightarrow{V_1} := (\overrightarrow{g} \circ \overrightarrow{p})\, \overrightarrow{p} \quad , \tag{7.1.34}$$

wobei $\overrightarrow{V_1}$ eine Projektion von $\overrightarrow{g}$ auf die Gerade des Vektors $\overrightarrow{p}$ ist, während $\overrightarrow{V_2}$ eine Projektion von $\overrightarrow{g}$ auf die Ebene senkrecht/orthogonal zu $\overrightarrow{p}$ ist.

Man betrachte ein orthonormales Koordinatensystem mit einer Basis, dessen letzter Vektor (Basisvektor) gleich $\overrightarrow{p}$ ist (Abbildung 7.1.3).
Sei V eine orthogonale Transformation, wobei gilt

$$\overrightarrow{g'} = (g'_1, \ldots, g'_r) = \overrightarrow{g}\, V \tag{7.1.35}$$

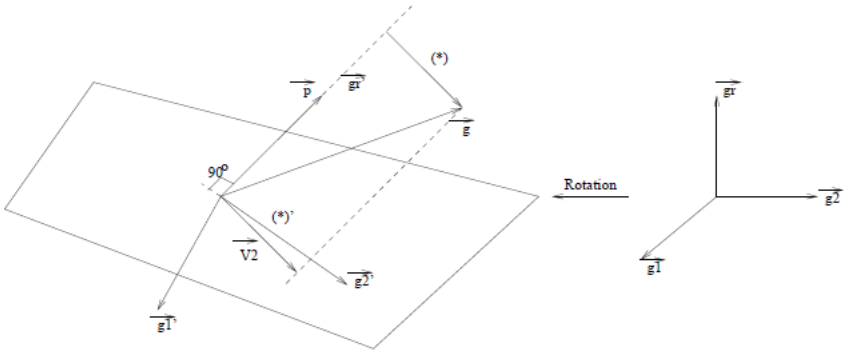

Abbildung 7.1.3: Rotation des Koordinatensystems [16]

Zu zeigen ist nun, dass die neuen Koordinatenvektoren $g'_1, \ldots, g'_r$ ebenfalls unabhängig standardnormalverteilt sind und der gleichen mehrdimensionalen Verteilung folgen wie $(g_1, \ldots, g_r)$.

Hierfür verwende man die Orthogonaltransformation:

$$\overrightarrow{g'} = \overrightarrow{g}\, V = (g_1, \ldots, g_r) V = (g'_1, \ldots, g'_r) \quad , \tag{7.1.36}$$

wobei gilt:

$$g'_s = \sum_{l=1}^{r} v_{ls}\, g_l \quad \forall s = 1, \ldots, r \tag{7.1.37}$$

Für $\overrightarrow{v_s} = (v_{1s}, \ldots, v_{rs})$ gilt:

$$\overrightarrow{v_s} \perp \overrightarrow{v_t} \quad \forall s, t = 1, \ldots, r \; ; \; s \neq t \tag{7.1.38}$$

$\overrightarrow{v_s}$ ist ein Spaltenvektor der Orthogonalmatrix V mit der Eigenschaft

$$\sum_{l=1}^{r} v_{ls}^2 = 1 \quad \text{, d.h. } |\overrightarrow{v_s}| = 1 \quad \forall s = 1, \ldots, r \tag{7.1.39}$$

Es gilt:

$$\begin{aligned}
\mathbb{E}\left[g_s'\right] &= \sum_{l=1}^{r} v_{ls}\,\mathbb{E}[g_l] = 0 \quad \text{und} \\
\operatorname{Var}(g_s') &= \mathbb{E}\left[(g_s')^2\right] - \left(\mathbb{E}\left[g_s'\right]\right)^2 \\
&= \mathbb{E}\left[(g_s')^2\right] \\
&= \mathbb{E}\left[\left(\sum_{l=1}^{r} v_{ls}\,g_l\right)^2\right] \\
&= \sum_{l=1}^{r} v_{ls}^2\,\mathbb{E}\left[g_l^2\right] \\
&= \sum_{l=1}^{r} v_{ls}^2 = 1
\end{aligned}$$

Damit wäre die folgende Verteilung gezeigt:

$$g_s' \sim \mathcal{N}(0,1) \quad \forall s = 1,\ldots,r \tag{7.1.40}$$

Man betrachte nun die Korrelation zwischen g_s' und g_t' für alle $s,t = 1,\ldots,r\ ;\ s \neq t$:

$$\begin{aligned}
\mathbb{E}\left[g_s'\,g_t'\right] &= \mathbb{E}\left[\left(\sum_{l=1}^{r} v_{ls}\,g_l'\right)\left(\sum_{l=1}^{r} v_{lt}\,g_l'\right)\right] \\
&= \sum_{l_1=1}^{r}\sum_{l_2=1}^{r} v_{l_1 s}\,v_{l_2 t}\,\mathbb{E}\left[g_{l_1}'\,g_{l_2}'\right] \\
&= \sum_{l=1}^{r} v_{ls}\,v_{lt}\,\mathbb{E}\left[(g_l')^2\right] \\
&= \sum_{l=1}^{r} v_{ls}\,v_{lt} \\
&= \overrightarrow{v_s} \circ \overrightarrow{v_t} = 0
\end{aligned}$$

Damit ist die Unkorreliertheit der Vektoren g_s' und g_t' gezeigt.

Man beweise, dass $(g_1',\ldots,g_r')$ die gleiche mehr- bzw. multidimensionale Verteilung wie $(g_1,\ldots,g_r)$ aufweist. Dies geschieht mit Hilfe der momenterzeugenden Funktion.

Gegeben sei ein r-dimensionaler Vektor $\overrightarrow{t} = (t_1, \ldots, t_r)$. Man bildet die momenterzeugende Funktion der unabhängigen Folge von Zufallsvariablen $\overrightarrow{g} = (g_1, \ldots, g_r)$:

$$\begin{aligned}
\varphi_{\overrightarrow{g}}(\overrightarrow{t}) &= \mathbb{E}\left[e^{\overrightarrow{g}\,\overrightarrow{t}^T}\right] \\
&= \mathbb{E}\left[e^{\sum_{l=1}^{r} t_l g_l}\right] \\
&= \prod_{l=1}^{r} \mathbb{E}\left[e^{t_l g_l}\right] \\
&= \prod_{l=1}^{r} e^{\frac{1}{2} t_l^2} \\
&= e^{\frac{1}{2}\sum_{l=1}^{r} t_l^2} \\
&= e^{\frac{1}{2}|t|^2}
\end{aligned}$$

Es ist bekannt, dass $\overrightarrow{g'}$ eine Ortogonalprojektion von $\overrightarrow{g}$ ist, d.h.

$$\overrightarrow{g'} = \overrightarrow{g}\,V \tag{7.1.41}$$

und für einen beliebigen Vektor $\overrightarrow{t} = (t_1, \ldots, t_r)$ gilt:

$$\sum_{l=1}^{r} t_l g_l' = (g_1', \ldots, g_r') \begin{pmatrix} t_1 \\ \vdots \\ t_r \end{pmatrix} \tag{7.1.42}$$

$$= \overrightarrow{g'}\,\overrightarrow{t}^T \tag{7.1.43}$$

$$= \overrightarrow{g}\,V\,\overrightarrow{t}^T \tag{7.1.44}$$

Unter Verwendung dieser Eigenschaft wird die momenterzeugende Funktion für $\overrightarrow{g'}$ gebildet:

$$\varphi_{\overrightarrow{g'}}(\overrightarrow{t}) = \mathbb{E}\left[e^{\overrightarrow{g'}\,\overrightarrow{t}^T}\right] \tag{7.1.45}$$

$$= \mathbb{E}\left[e^{\sum_{l=1}^{r} t_l g_l'}\right] \tag{7.1.46}$$

$$= \mathbb{E}\left[e^{\overrightarrow{g}\,V\,\overrightarrow{t}^T}\right] \tag{7.1.47}$$

$$= \mathbb{E}\left[e^{\overrightarrow{g}\,(\overrightarrow{t}\,V^T)^T}\right] \tag{7.1.48}$$

Nun erhält man eine momenterzeugende Funktion von $\overrightarrow{g}$ im Punkt $\overrightarrow{t}V^T$, wobei gilt:

$$\varphi_{\overrightarrow{g}}(\overrightarrow{t}V^T) = e^{\frac{1}{2}\left|\overrightarrow{t}V^T\right|^2} \tag{7.1.49}$$

$$= e^{\frac{1}{2}\left|\overrightarrow{t}\right|^2} \tag{7.1.50}$$

Im letzten Schritt der Gleichung benutzt man eine Eigenschaft der Orthogonalprojektion, welche besagt, dass die Länge von Vektoren beibehalten wird.

Damit wurde gezeigt, dass für die momenterzeugenden Funktionen Gleichheit gilt:

$$\varphi_{\overrightarrow{g}}(\overrightarrow{t}) = \varphi_{\overrightarrow{g'}}(\overrightarrow{t}) \tag{7.1.51}$$

An dieser Stelle ist festzuhalten, dass $(g'_1, \ldots, g'_r)$ die gleiche mehr- bzw. multidimensionale Verteilung hat wie $(g_1, \ldots, g_r)$.

Durch die Wahl der Basis des neuen orthonormalen Koordinatensystems wird sichergestellt, dass der Vektor (7.1.33)

$$\overrightarrow{V_2} = \overrightarrow{g} - (\overrightarrow{g} \circ \overrightarrow{p})\,\overrightarrow{p}$$

nach der Orthogonaltransformation die Koordinaten

$$(g'_1, \ldots, g'_{\hat{k}-1}, 0) \tag{7.1.52}$$

besitzt.

Hiermit ist die Behauptung aus (7.1.23) bewiesen, da gilt:

$$\sum_{j=1}^{\hat{k}} Z_j^2 = \sum_{j=1}^{\hat{k}} (j\text{-te Koordinate})^2 \tag{7.1.53}$$

$$= (g'_1)^2 + \cdots + (g_{\hat{k}-1})^2 \sim \chi^2_{\hat{k}-1} \tag{7.1.54}$$

7.2 Vergleich der Gütemaße Modularität und Kontingenzkoeffizient

Als Berechnungsgrundlage des Kontingenzkoeffizienten dient die Häufigkeitsmatrix, wobei sich die Abweichung der tatsächlichen von den erwarteten Einträgen in den Ausprägungen des Maßes widerspiegelt. Dieses Prinzip ist äquivalent zum Gütemaß der Modularität, welche in Kapitel 6.2 eingeführt wurde.
Im Folgenden sollen deshalb die beiden Maße sowie ihre zugrundeliegenden Matrizen verglichen werden.

Beide Matrizen sind quadratisch und besitzen die Dimension $k \times k$, wobei k die Clusterzahl beschreibt. Die Matrixeinträge des Graph Clustering beschreiben die Kantengewichte inner- und interdirektional bezüglich ihrer Clusterverknüpfungen. Die Matrix $\boldsymbol{e}$ ist symmetrisch, da gilt:

$$e_{ij} = e_{ji} \qquad \forall i, j \in \{1, \ldots, k\},\ i \neq j \tag{7.2.1}$$

Diese Eigenschaft führt zur Gleichheit der Zeilen- und Spaltensummen (Kapitel 6.2):

$$\begin{aligned} a_i &= \sum_{j=1}^{k} e_{ij} \qquad \forall i = 1, \ldots, k \\ b_j &= \sum_{i=1}^{k} e_{ij} \qquad \forall j = 1, \ldots, k \\ a_i &= b_j \qquad \forall i, j \in \{1, \ldots, k\},\ i = j \end{aligned}$$

Es wird eine hohe Konzentration der Spureinträge angestrebt, d.h. die Einträge auf der Diagonalen der Matrix sollen addiert einen möglichst großen Anteil an der Gesamtsumme der Kantengewichte darstellen. Weiterhin sollten die k Einträge der Spur im besten Falle gleichverteilt sein. Folglich werden auf den Einträgen außerhalb der Spur geringe Ausprägungen erwartet. Der optimale Fall, respektive die maximale Modularität (vgl. 6.2.9) $Q^* = 1$ wird dann erreicht, wenn:

$$e_{ii} = \frac{1}{k} \qquad \forall i = 1, \ldots, k \tag{7.2.2}$$

$$e_{ij} = 0 \qquad \forall i, j \in \{1, \ldots, k\},\ i \neq j \tag{7.2.3}$$

Die erwarteten Einträge der Matrizen berechnen sich durch Multiplikation der Zeilen- und Spaltensummen. Ihre Ausprägungen werden unter der Prämisse ermittelt, dass die Kantengewichte (Graph Clustering) bzw. die Häufigkeiten (Clusterzusammenführung) in Zeile und Spalte gleichmäßig verteilt werden.

$$\hat{e}_{ij} = a_i\, b_j \qquad \forall i, j \in \{1, \ldots, k\} \quad \text{Matrix } \boldsymbol{e} \text{ (Graph Clustering)} \tag{7.2.4}$$

$$\hat{u}_j = \frac{u_{k_{r*}}\, u_{k_{*s}}}{M} \qquad j = 1, \ldots, \hat{k} \quad \text{Häufigkeitsmatrix } \boldsymbol{U} \tag{7.2.5}$$

Im Kontext der Clusterzusammenführung wird das Ziel verfolgt, dass sich möglichst große Anteile der Objektzahl M auf wenige Einträge pro Zeile bzw. Spalte der Häufigkeitsmatrix verteilen. Im Umkehrschluß würde sich eine zufällige (Gleich-)Verteilung der Anteile auf sämtliche Einträge einer Zeile oder Spalte negativ auf das Gütemaß auswirken.

Entsprechend erhalte man eine Optimallösung mit $\chi^2 = M(k-1)$, was einen maximalen korrigierten Kontingenzkoeffizienten K^* von 1 bedeuten würde, wenn:

$$\forall k_r = 1,\ldots,k \quad \exists \text{ genau ein } \tilde{k}_s \text{ mit } u_{k_r,\tilde{k}_s} > 0,\ u_{k_r,\tilde{k}_s} = u_{k_r,*} \tag{7.2.6}$$

und

$$\forall k_s = 1,\ldots,k \quad \exists \text{ genau ein } \tilde{k}_r \text{ mit } u_{\tilde{k}_r,k_s} > 0,\ u_{\tilde{k}_r,k_s} = u_{*,k_s} \tag{7.2.7}$$

Im Vergleich zu einer Optimallösung des Graph Clustering ist es nicht zwingend erforderlich, dass die k positiven Einträge die Spur der Matrix bilden. Ebenso wenig müssen sie identische Werte von $\frac{1}{k}$ vorweisen. Sollten diese beiden Bedingungen allerdings erfüllt sein, so wäre der Kontingenzkoeffizient der Clusterzusammenführung selbstverständlich weiterhin maximal.
Das bedeutet, überführt man das Matrixmuster einer Optimallösung ($Q^* = 1$) im Sinne des Graph Clustering in sein Äquivalent einer Häufigkeitsmatrix, dann erhält man ebenfalls eine Optimallösung ($K^* = 1$).

Die Eigenschaften der Instanz dieser Arbeit schränken die Zahl möglicher Matrixstrukturen sowohl des Graph Clustering als auch der Clusterzusammenführung ein.
Die Nebenbedingungen der Clusteranalysen nach k-Means und k-Modes (5.5.2 und 5.4.2) verweisen jeweils auf eine Objektzahl von größer gleich 1 pro Cluster. Für die Häufigkeitsmatrix bedeutet dies, dass gilt:

$$u_{*,k_s} > 0 \qquad \forall k_s = 1,\ldots,k \quad \text{und} \tag{7.2.8}$$

$$u_{k_r,*} > 0 \qquad \forall k_r = 1,\ldots,k \quad . \tag{7.2.9}$$

Dies hat folgende Auswirkungen auf die Kontingenzkoeffizienten:

$$\chi^2 > 0 \tag{7.2.10}$$

$$K > 0 \tag{7.2.11}$$

$$K^* > 0 \tag{7.2.12}$$

Auf Seiten des Graph Clustering ist festzuhalten, dass alle Objekte paarweise untereinander verknüpft sind und ein Kantengewicht größer 0 vorweisen. Für die Einträge der Matrix $\boldsymbol{e}$ gilt deshalb:

$$e_{ij} > 0 \qquad \forall i,j \in \{1,\ldots,k\},\ i \neq j \tag{7.2.13}$$

$$e_{ii} = 0 \qquad \text{falls für Cluster } i \text{ gilt: } |V_C| = 1,\ i \in \{1,\ldots,k\} \tag{7.2.14}$$

Die Spur der Matrix kann deshalb nicht alle Anteile der Kantengewichte beinhalten. Es gilt $Tr(\boldsymbol{e}) < 1$. Die Ausprägung der Modularität kann daher nicht

maximal werden.

$$Q < \frac{k-1}{k} \tag{7.2.15}$$

$$Q^* < 1 \tag{7.2.16}$$

7.3 Umsetzung in KNIME

In Kapitel 5.6 wurden der enorme Rechenaufwand und die große Datenmenge bei der Durchführung der Clusteranalyse in KNIME dargelegt. Dies hat selbstverständlich Auswirkungen auf die Umsetzung der Clusterzusammenführung. So erfordert die große Menge an generierten Clusterkombinationen einen Workaround bei der Speicherung dieser, indem für jedes $k \in [3;10]$, $k \in \mathbb{N}$ eine separate Outputtabelle erzeugt wird (Abbildung 7.3.1).

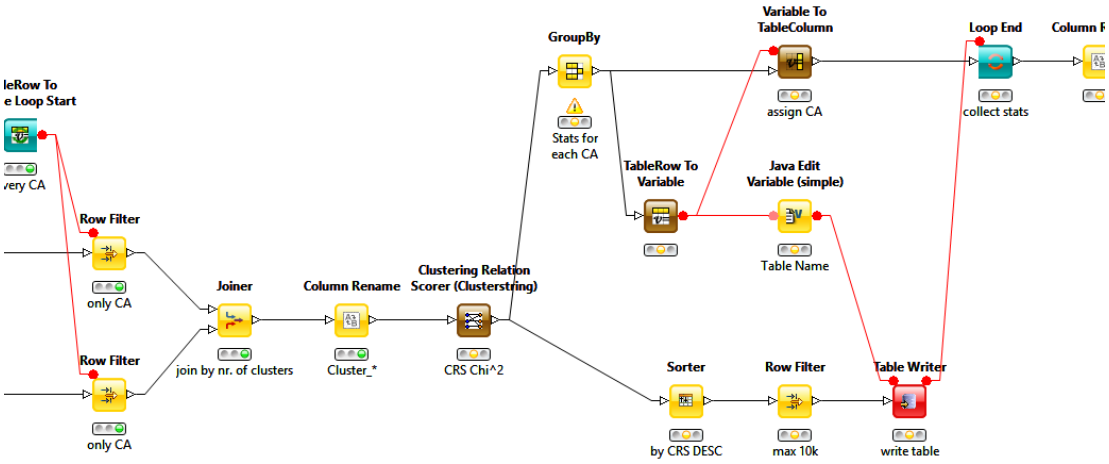

Abbildung 7.3.1: Routen der Clusterzusammenführung und temporäres Speichern [KNIME]

Die Verkettung der Clusterlösungen nach Weichenstamm- und Messdaten erfolgt mit Hilfe eines Join-Knotens, wobei die Clusterzahl k als Übereinstimmungsmerkmal festgelegt wird (Abbildung 7.3.2).

Die anschließende Berechnung des korrigierten Kontingenzkoeffizienten zur Bewertung einer Clusterkombination erfolgt mit Hilfe eines eigens programmierten Moduls (Abbildung 7.3.3), womit sich ein zeit- und rechenaufwendiges

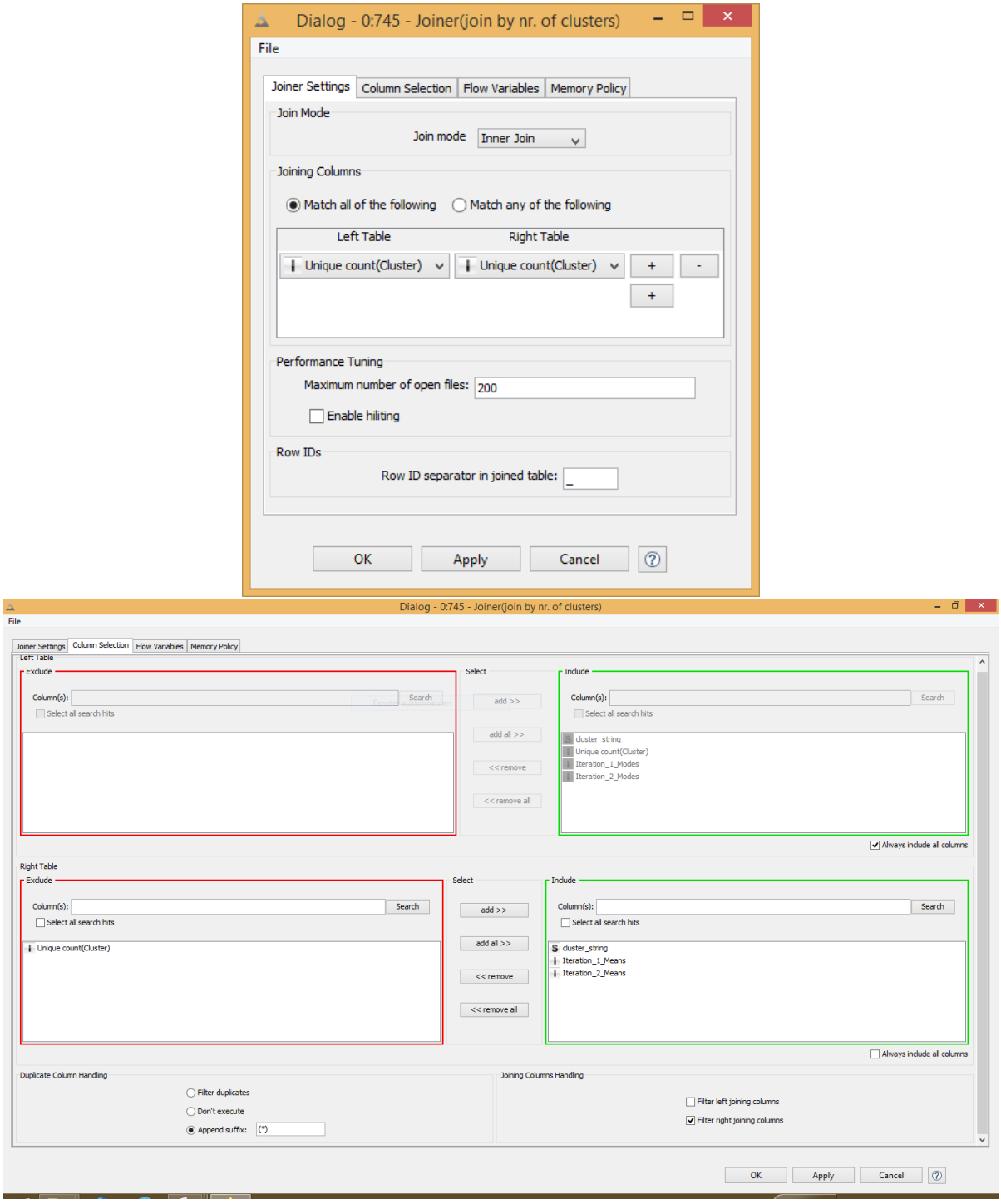

Abbildung 7.3.2: Konfiguration des *Join*-Moduls [KNIME]

Durchlaufen einer Route erübrigt. Dabei werden die Rechenoperationen innerhalb eines Knotens abgewickelt und der Wert des Gütemaßes einer zusätzlichen Spalte der Tabelle hinzugefügt (Quellcode Quellcode A.1, A.3, A.2).

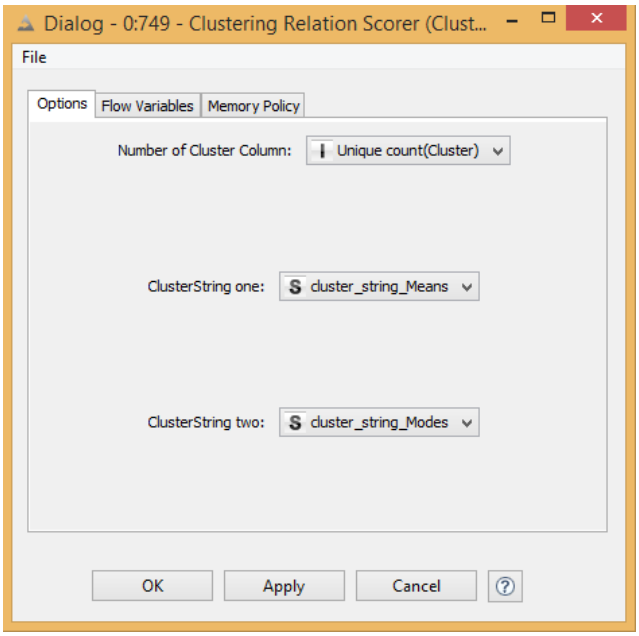

Abbildung 7.3.3: Konfiguration des Moduls zur Berechnung des Kontingenzkoeffizienten [KNIME]

Die Menge an erzeugten Daten im Zuge der Clusterzusammenführung ist immens, so dass ein Zwischenspeichern nach dem Durchlaufen einer Clusterzahl k notwendig erscheint. Da KNIME kein schrittweises Durchlaufen eines Kreislaufs mit Speicherung der gesammelten Daten und Leeren des Caches bietet, wurde ein Lösungsansatz per Variablenfluss (Abbildung 7.3.1) gewählt.

Tabelle 7.3.1 zeigt die Entwicklung des Gütemaßes über die Clusterzusammenführungen der 8 Clusterzahlen sowie die Anzahl der analysierten Lösungen.

		Clusterzahl k							
		3	4	5	6	7	8	9	10
eindeutige Kombinationen		2019325	3231206	3476964	3497856	3434093	3488747	3388548	3388548
Minimum	Clustering Relation Scorer	0,013	0,125	0,228	0,348	0,401	0,499	0,578	0,604
Mittelwert		0,434	0,538	0,600	0,659	0,703	0,741	0,775	0,799
Maximum		0,904	0,899	0,903	0,906	0,910	0,907	0,928	0,936
Standardabweichung		0,127	0,089	0,069	0,056	0,049	0,042	0,036	0,032
TOP 10000									
Minimum	Clustering Relation Scorer	0,693	0,733	0,782	0,822	0,846	0,859	0,875	0,886
Mittelwert		0,715	0,761	0,808	0,842	0,859	0,870	0,883	0,893
Maximum		0,904	0,899	0,903	0,906	0,910	0,907	0,928	0,936
Standardabweichung		0,018	0,027	0,024	0,013	0,010	0,009	0,007	0,006

Tabelle 7.3.1: Resultate des Clustering Relation Scorers

Wie in der Tabelle angedeutet, sollen für die nun folgende Störungsanalyse die besten 10000 Clusterzusammenführungen pro Clusterzahl einer tieferen störungsspezifischen Untersuchung unterzogen werden. Insgesamt verbleiben damit 80000 Kombinationen, die auf Grund eines korrigierten Kontingenzkoeffizienten von $Q^* > 0,693$ ähnliche Objektstrukturen in den Partitionen nach Mess- und Weichenstammdaten vorweisen.

Kapitel 8

Störungsanalyse

Neben der umfangreichen Messdatenreihe und den Ausprägungen der Weichenstammdaten liegt dieser Arbeit eine Sammlung von Störungsdaten als Input vor. Diese Daten wurden direkt von den Mitarbeitern der Instandsetzung beim Auftreten einer Weichenstörung bzw. des Ausfalls einer Weichenanlage aufgenommen. Sie beinhalten keinerlei Informationen des Diagnosesystems SIDIS W und treten völlig unabhängig von der Funktion des Systems auf. Weiterhin sind sie - wie die Elemente der Messdatenreihe - genau einem Objekt zugeordnet. Daher war es möglich die Störungsdaten nahtlos in die Objektzustände zu integrieren.
Vor dieser Eingliederung ist es notwendig die Datenreihe der Störungen von irrelevanten bzw. unbrauchbaren Elementen zu bereinigen. Dies können Duplikate sein, Datensätze, welche sich durch fehlende Informationen auszeichnen oder Störungselemente von Objekten, die in einem vordefinierten Zeitraum vor dem Auftreten der Störung keine Messdaten aufweisen.

Ein wichtiges Attribut jeder Störung ist die Priorität mit Ausprägungen $\{1, \ldots, 7\}$, wobei dem Wert 1 die größte Brisanz zugeordnet wird („Unfall mit Todesfolge nach Weichenausfall“) und im Beobachtungszeitraum nicht verzeichnet ist. Störungen der Priorität 7 wurden auf Grund fehlender Interpretations- und Reproduktionsmöglichkeit herausgefiltert. An dieser Stelle wurde eine Klassifizierung durchgeführt, indem die übrigen 5 Prioritätsstufen eindeutig drei Klassen zugeteilt wurden (siehe Tabelle 8.0.1).

Prioritätsklasse	Prioritätsstufen
1	{2,3}
2	{4,5}
3	{6}

Tabelle 8.0.1: Prioritätsklassen

Bei der Störungsanalyse soll die Entwicklung der Messwerte vor dem Störungseintritt untersucht werden. Vorab wird eine Bewertung der Partitionen bezüglich der Prioritätsklassenverteilung durchgeführt, um besonders aussagekräftige Resultate für die weitere Analyse zu filtern.

8.1 Störungsinformationen und Störungszustand

Bei jedem Element der Störungsdatenreihe sind folgende Parameter von Bedeutung:

(i) Akronym $\boldsymbol{W_m}$: Objekt $m = 1, \ldots, M$, $M = 111$

(ii) Akronym $\boldsymbol{n_S}$: fortlaufende Störungsnummer für ein Objekt

(iii) Akronym $\boldsymbol{t}$: Zeitpunkt der Störung als Zeitstempel, $f_{m,n_S,t}$

(iv) Akronym $\boldsymbol{d}$: Dauer zwischen Störungseintritt und -behebung [sec], $f_{m,n_S,d}$

(v) Akronym $\boldsymbol{p}$: Prioritätsklasse, $f_{m,n_S,p}$

Die Störungsnummer n_S ist durch die Störungszahl N_m^S (Kapitel 3.3) beschränkt.

$$n_S \in \{1, \ldots, N_m^S\} \quad \text{falls } N_m^S \geq 1. \tag{8.1.1}$$

F_{m,n_S} beschreibt die n_S-te Störung des Objekts W_m und beinhaltet sämtliche Informationen bzw. Eigenschaften des Störungsvorfalls.

$$F_{m,n_S} = (f_{m,n_S,t}, f_{m,n_S,d}, f_{m,n_S,p}) \qquad \forall m = 1, \ldots, M;\ n_S \in [1, N_m^S] \tag{8.1.2}$$

Wie bereits eingangs des Kapitels erwähnt, sollen die Weichenzustände $\boldsymbol{X}_{m,n}$ mit den Störungsinformationen angereichert werden. Hierfür wird die Messdatenreihe eines jeden Objekts, welche die Menge der Weichenumläufe beschreibt um die Störungsvorfälle erweitert, indem der Störungszustand Δ als zusätzliche Eigenschaft in der Attributmenge berücksichtigt wird.
Um dies zu bewerkstelligen wird ein erweiterter Zustandsraum (4.3.7) eingeführt.

Die Zeitpunkte der Störungen werden als Stoppzeiten [13, S.196] bezeichnet, was eine Unterbrechung der Funktionalität des Objekts implizieren soll.

$$\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{N_m^S}) \tag{8.1.3}$$

$\boldsymbol{\tau}$ sei eine Familie von Stoppzeiten bzw. Stoppzeitpunkte des Stochastischen Prozesses $\boldsymbol{X_m}$ (4.3.11). Der Messdaten-Zustand $X_{m,\tau_{n_S},A}$ zum Zeitpunkt einer Störung ergibt sich wie folgt:

$$\forall m \text{ mit } N_m^S \geq 1 \; \exists$$
$$X_{m,\tau_{n_S},A} = \left(x_{m,\tau_{n_S},A_i} : i \in I^A,\; x_{m,\tau_{n_S},A_i} \in DOM(A_i)\right) \times \Delta \qquad \forall n_S \in [1, N_m^S]$$

Die Ermittlung der Parameterwerte im kritischen Zustand

$$x_{m,\tau_{n_S},A_i} : i \in I^A,\; x_{m,\tau_{n_S},A_i} \in DOM(A_i) \tag{8.1.4}$$

soll im folgenden Kapitel 8.3 erläutert werden.

An dieser Stelle sei erwähnt, dass nicht jedes Objekt im Beobachtungszeitraum eine Störung aufweist, weshalb für zutreffende Objekte gilt:

$$\boldsymbol{\tau} = (\varnothing) \tag{8.1.5}$$

Alle Weichen W_m mit $m = 1, \ldots, M$, die sich durch $N_m^S \geq 1$ auszeichnen, werden als Störungsobjekte bezeichnet.

Weichenzustände, welche keiner Störung zugeordnet werden bleiben selbstverständlich von den „benachbarten“ Ausfällen unberührt.

8.2 Analyse der Störungspriorität

Die Einführung von Prioritätsklassen basiert auf der ähnlichen Aussagekraft der Prioritätsstufen. Ziel soll es sein, dass Störungen einer Prioritätsklasse zu einem großen Prozentsatz genau einem Cluster zugeteilt werden können. Es ist daher notwendig die Objekte zum Einen auf die Anzahl ihrer Störungen und zum Anderen auf die zugehörigen Prioritätsklassen zu untersuchen. Anschließend soll festgestellt werden, ob im Zuge der Clusteranalyse ähnliche Objektmuster bezüglich ihrer Störungen zusammengeführt werden.

Folgende Umstände und Einschränkungen sind bei der Analyse der Störungsprioritätenverteilung zu berücksichtigen:

(i) Von 39 Störungsobjekten können 34 genau einer Prioritätsklasse zugeteilt werden.
Die fünf übrigen werden von der Analyse ausgeschlossen.

(ii) Prioritätsklasse 3 tritt lediglich einmal auf und das zugehörige Objekt wird unter (i) ausgeschlossen.
Lediglich die Verteilung der Klassen 1 (6 Objekte) und 2 (28 Objekte) ist von Interesse.

(iii) Für jedes Cluster (nach k-Means) wird die Konzentration der beiden Prioritätsklassen bestimmt, d.h. der prozentuale Anteil der Störungsobjekte einer Klasse im entsprechenden Cluster wird berechnet.
Man beschränkt sich in dem Zusammenhang auf die Partitionen nach dem k-Means Verfahren.

(iv) Für die übrigen Prioritätsklassen sollen die Zufallsvariablen Q_1 und Q_2 die Konzentration der Störungsobjekte in einem Cluster repräsentieren. Anschließend soll die Streuung ermittelt werden.

(v) Ein Cluster darf nicht die maximale Konzentration für beide Klassen vorweisen.

Tabelle 8.2.1: Prioritäten der Störungsdaten

Die zusammenfassende Bewertung erfolgt durch Mittlung der beiden Varianzen 8.2.1. Für eine fortführende Analyse des Clusterergebnisses muss der Term einen Wert überschreiten. Diese Grenze enthält die Clusterzahl k als Parameter und wird mit zunehmendem k herabgesetzt.

$$\frac{Var(CC_1) + Var(CC_2)}{2} \overset{!}{>} 0,03^{\left(\frac{k}{6,5}\right)^{0,3}} \tag{8.2.1}$$

Es gilt:

$$Var(CC_1), Var(CC_2) < 1$$
$$Var(CC_1) + Var(CC_2) < 2$$

Tabelle 8.2.2 zeigt eine Statistik über die Ergebnisse der Filterung. Ausgehend von 1928 Lösungen der Clusteranalyse nach k-Means „überstehen“ 438 Lösungen den Test auf Prioritätsklassenverteilung.

	Clusterzahl k								$\sum$
	3	4	5	6	7	8	9	10	
Anzahl Ergebnisse Input	454	352	79	141	153	219	241	289	1928
Mittleres Varianzmittel	0,0775	0,05822	0,0481	0,0345	0,0257	0,0214	0,0179	0,0158	
Ausschluß	388	269	39	86	107	164	179	213	1445
Anzahl Ergebnisse Output	66	83	40	55	46	55	62	76	483

Tabelle 8.2.2: Ergebnisse der Filterung nach Verteilung der Störungsprioritäten

Die Bewertung von Clustern einer Lösung auf Grundlage der Verteilung ihrer zugeordneten Störungen ist an dieser Stelle abgeschlossen. Im Folgenden soll nun analysiert werden, inwiefern Störungen im Vorlauf durch auffällige Zustandsänderungen erkannt werden können.

8.3 Störungsrelevante Messdaten

In Kapitel 8.1 wurden die Eigenschaften von Störungen eingeführt und deren Vorkommen in die Aufzeichnungsmenge der Weichenobjekte eingegliedert. Um die Konsistenz dieser Menge sicherzustellen, ist es notwendig den Störungszuständen passende Parameterwerte (8.1.4) zuzuweisen.

Da im Rahmen der Störungsaufzeichnung kein neues Messdatenelement erfasst wird (es findet nicht zwingend zum Auftrittszeitpunkt der Störung ein Weichenumlauf statt), ist es notwendig plausible Messwerte für den Störungszustand (8.1.4) zu generieren. Dies geschieht durch Aggregation einer Menge vorangegangener Messwerte, die als störungsrelevant deklariert werden.
Der Aggregationsoperation liegt folgende Gewichtungsfunktion zu Grunde (Abbildung 8.3.1):

$$w_{m,n_S}(n) = \begin{cases} e^{\frac{0,66}{\left(\frac{f_{m,n_S,t}-x^A_{m,n,15}}{86400}\right)^{0,65}}} & \text{wenn} \quad f_{m,n_S,t} - 5d \leq x^A_{m,n,15} < f_{m,n_S,t} - \frac{1}{12}d \\ e^{\frac{0,66}{\left(\frac{1}{12}\right)^{0,65}}} = 27,63 & \text{wenn} \quad f_{m,n_S,t} - \frac{1}{12}d \leq x^A_{m,n,15} \leq f_{m,n_S,t} \\ 0 & \text{sonst} \end{cases}$$
$$\forall n \in [1; N_m], n \notin \boldsymbol{\tau};\ f_{m,n_S,t}, x^A_{m,n,15} \text{ in [sec]} \tag{8.3.1}$$

Die Differenz $f_{m,n_S,t} - x^A_{m,n,15}$ ist hier nicht als „time to failure“ zu verstehen, sondern als Zeitdifferenz zwischen dem n-ten Weichenumlauf und der darauffolgenden n_S-ten Störung. Je kleiner diese Zeitdifferenz ausfällt, desto stärker

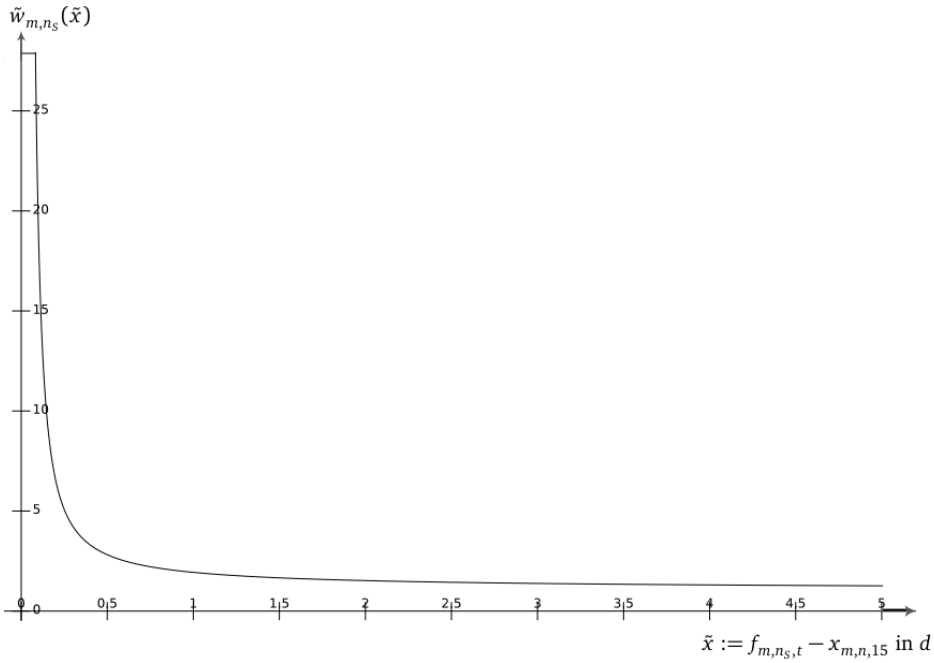

Abbildung 8.3.1: Plot der Gewichtungsfunktion angelehnt an 8.3.1

werden die Zustandswerte bei der Aggregation störungsrelevanter Daten berücksichtigt.

Die Einführung der Zeitschranken von 5 Tagen und $\frac{1}{12}d = 2$ Stunden ist in der operativen Arbeit der Instandhaltung begründet. So werden Wartungsarbeiten maximal 5 Tage im Voraus „fein“ geplant und Ad-Hoc-Maßnahmen können innerhalb weniger als 2 Stunden durchgeführt werden (Notfallplanung). Diese Umstände der Arbeit in den Stellwerken der DB AG beruhen auf Erfahrungswerte in der Instandhaltung und technische Funktionseigenschaften des Assets Eisenbahnweiche.

Φ_{m,n_S} sei die Menge der störungsrelevanten und somit kritischen Messdaten.

$$\Phi_{m,n_S} = \left\{ X^A_{m,n} : w_{m,n_S}(n) > 0;\, n \in [1; N_m] \right\} \quad \forall m = 1, \ldots, M;\, n_S \in [1, N^S_m] \tag{8.3.2}$$

Entsprechend werden die Messdaten unter Multiplikation mit ihren Gewich-

ten aggregiert, um störungscharakterisierende Messwerte x_{m,τ_{n_S},A_i} zu erhalten:

$$\forall m : N_m^S \geq 1,\, n_S \in [1, N_m^S] :$$

$$x_{m,\tau_{n_S},A_i} := \frac{1}{\sum_{n=1}^{N_m} w_{m,n_S}(n)} \sum_{n=1}^{N_m} w_{m,n_S}(n)\, x_{m,n,A_i} \qquad \forall i \in I^A \backslash \{15\} \tag{8.3.3}$$

$$x_{m,\tau_{n_S},A_{15}} := f_{m,n_S,t} \tag{8.3.4}$$

Letztendlich wurden die Störungsdaten final analysiert und es wurden jeder Störung charakterisierende Messwerte zugeordnet.

$$\forall\, W_m :\ m = 1, \ldots, M;\ N_m^S > 0 :$$
$$\forall F_{m,n_S} : n_S \in [1; N_m^S] \quad \exists\, x_{m,\tau_{n_S},A_i} \quad \forall i \in I^A,$$
sodass gilt:

$$X_{m,\tau_{n_S},A} = (x_{m,\tau_{n_S},A_i} : i \in I^A) \times \Delta \quad , \tag{8.3.5}$$

$$X_{m,\tau_{n_S},A} \text{ ist } \left([\prod_{i \in I^A} \Omega_{A_i}] \times \Omega_S \,,\, [\bigotimes_{i \in I^A} \mathscr{A}_{A_i})] \otimes \mathscr{A}_S \right) \text{-messbar} \qquad \forall n_S \in [1, N_m^S] \tag{8.3.6}$$

8.4 Störungsanalyse auf Clusterebene

Die Zuordnung der Störungen ihren Objekten sowie die Berücksichtigung, Berechnung und Einbettung des Störungszustands stellen wichtige vorbereitende Maßnahmen der Störungsanalyse dar. Weiterführend ist es notwendig für jedes Cluster störungscharakterisierende Messwerte zu berechnen. An dieser Stelle sollen die Segmentierungen nach Messdaten ins Zentrum der Analyse rücken, wohl wissend dass sich die Objektstrukturen in den Produktdaten ähneln (Kapitel 7.3).
Rückblickend auf die Filterung der Lösungen im vorangegangenen Kapitel 8.3 kann davon ausgegangen werden, dass sich die Störungen in einem Cluster insofern gleichen, als dass sie häufig in ihrer Prioritätsklasse übereinstimmen.

Ziel ist ein Vergleich der angesprochenen störungscharakterisierenden Variablenwerte der Messdaten (Zufallsvariable $Z_{2,i} \sim \left(\mu_{2,i}, \sigma_{2,i}^2\right)$) mit der größeren Menge an unkritischen Ausprägungen innerhalb eines Clusters. Letzterer Typ wird unterteilt in Variablenwerte der Störungsobjekte (Zufallsvariable $Z_{1,i} \sim \left(\mu_{1,i}, \sigma_{1,i}^2\right)$) und derjenigen Objekte ohne Störung ($N_m^s = 0$) (Zufallsvariable $Z_{0,i} \sim \left(\mu_{0,i}, \sigma_{0,i}^2\right)$).
Demnach werden sämtliche Aufzeichnungen der Objekte eines Clusters eindeutig einem von drei Tupeln zugeordnet. Dabei wird für die Zufallsvariablen der

Zustandstupel die Normalverteilung angenommen. Die Konfiguration von Clustern kann abweichen, vorausgesetzt die Objektgruppen bestehen aus keinerlei oder ausschließlich Störungsobjekten. In beiden Fällen wäre eine Unterteilung in lediglich zwei Tupeln notwendig.

$$N_{C_l}^{S} = \sum_{m=1}^{M} r_{m,l}\, I(N_m^S > 0) \qquad \forall l = 1,\ldots,k \tag{8.4.1}$$

$$N_{C_l}^{\bar{S}} = \sum_{m=1}^{M} r_{m,l}\, I(N_m^S = 0) \qquad \forall l = 1,\ldots,k \tag{8.4.2}$$

$$\forall\, C_l \quad l = 1,\ldots,k, \quad \forall i \in I:$$

$$\mu_{2,i} = \frac{1}{\sum_{m=1}^{M} r_{m,l}\, N_m^S} \sum_{m=1}^{M} r_{m,l}\, I(N_m^S > 0) \left(\sum_{n_S=1}^{N_m^S} x_{m,\tau_{n_S},i} \right)$$

$$N_{C_l}^{S} > 0;\ \forall i \in I^A,\ l = 1,\ldots,k \tag{8.4.3}$$

$$\mu_{1,i} = \frac{1}{N_{C_l}^{S}} \sum_{m=1}^{M} r_{m,l}\, \frac{I(N_m^S > 0)}{N_m - N_m^S} \left(\sum_{n=1}^{N_m} x_{m,n,i}\, I(n \notin \boldsymbol{\tau}) \right)$$

$$N_{C_l}^{S} > 0;\ \forall i \in I^A,\ l = 1,\ldots,k \tag{8.4.4}$$

$$\mu_{0,i} = \frac{1}{N_{C_l}^{\bar{S}}} \sum_{m=1}^{M} r_{m,l}\, \frac{I(N_m^S = 0)}{N_m} \left(\sum_{n=1}^{N_m} x_{m,n,i} \right)$$

$$N_{C_l}^{\bar{S}} > 0;\ \forall i \in I^A,\ l = 1,\ldots,k \tag{8.4.5}$$

Bei der Berechnung von $\mu_{2,i}$ wird im Falle von mehreren Störungen einer Weiche W_m ($N_m^S > 1$) auf eine Aggregation auf Objektebene verzichtet. Demnach sind alle Störungen gleichwertig, die dem Cluster C_l zugehörig sind.

Die definierten Zufallsvariablen spiegeln drei charakterisierende Größen eines Clusters wider. Im weiteren Verlauf sollen Statistiken für einen Vergleich der Zufallsvariablen eingeführt werden.

Wünschenswert wäre eine messbare Zustandsänderung eines Störungsobjekts innerhalb des kritischen Zeitraums vor einer Störung, d.h. der Vergleich störungsrelevanter Daten mit unkritischen Daten der Störungsobjekte steht im Zentrum der Analyse.
Im Gegenzug wird eine hohe Übereinstimmung unkritischer Werte angestrebt. Dies impliziert bereits eine optimale Lösung nach k-Means.

Als statistische Methode der Gegenüberstellung zweier Zufallsvariablen, unter der Annahme, dass beide unabhängig normalverteilt sind und die charakteristischen Variablenwerte eines Clusters widerspiegeln wird der *Zweistichproben-Gauß-Test* [8, S.292f] herangezogen.

$$\forall\ C_l \text{ mit } l = 1, \ldots, k, \quad \forall i \in I:$$

$$u_{1,i} := \frac{\mu_{0,i} - \mu_{1,i}}{\sqrt{\frac{\sigma_{0,i}}{N^{\tilde{S}}_{C_l}} + \frac{\sigma_{1,i}}{N^{S}_{C_l}}}} \tag{8.4.6}$$

$$u_{2,i} := \frac{\mu_{0,i} - \mu_{2,i}}{\sqrt{\frac{\sigma_{0,i}}{N^{\tilde{S}}_{C_l}} + \frac{\sigma_{2,i}}{\sum_{m=1}^{M} r_{m,l} N^{S}_{m}}}} \tag{8.4.7}$$

$$u_{3,i} := \frac{\mu_{1,i} - \mu_{2,i}}{\sqrt{\frac{\sigma_{1,i}}{N^{S}_{C_l}} + \frac{\sigma_{2,i}}{\sum_{m=1}^{M} r_{m,l} N^{S}_{m}}}} \tag{8.4.8}$$

Rückblickend auf die Forderungen, welche eingangs dieses Kapitels aufgestellt wurden, können diese nun aufgegriffen und formal wiedergegeben werden. So soll für $u_{3,i}$ ein hoher Wert und für $u_{1,i}$ ein niedriger Wert angestrebt werden.

Im Zuge der Bewertung einer jeden Partition durch Berechnung der Statistiken, soll die Performance der einzelnen Parameter evaluiert werden. Hierfür werden zunächst für jedes Cluster eines Resultats die Statistiken 8.4.6 und 8.4.8 berechnet, um anschließend ergebnisübergreifend die Streuung und damit die Aussagekraft der Messwertparameter zu ermitteln.

Es wird an dieser Stelle ein Hypothesentest zur Zustandsähnlichkeit eingeführt:

$$\forall i \in I:$$

$$H_0 : \mu_{0,i} = \mu_{1,i} \qquad H_1 : \mu_{0,i} \neq \mu_{1,i} \tag{8.4.9}$$

zum Signifikanzniveau $\alpha = 0.10$.
Entscheidungsregel: Ablehnung von H_0, wenn:

$$|u_{1,i}| > \phi\left(1 - \frac{\alpha}{2}\right) = 1,64 \tag{8.4.10}$$

Variable	**Attribut**	Clusterzahl k								∅
		3	4	5	6	7	8	9	10	
A_1	Pleer	0,7020	0,6536	0,6300	0,6212	0,6180	0,6114	0,5878	0,5882	0,6265
A_2	Plauf1	0,6717	0,6446	0,5950	0,6273	0,6025	0,6250	0,5806	0,5829	0,6162
A_3	Plauf2	0,6313	0,4940	0,5700	0,6091	0,5963	0,6295	0,6093	0,6237	0,5954
A_4	Pverschl	0,6717	0,6416	0,5900	0,6394	0,6211	0,6432	0,6254	0,6276	0,6325
A_5	PverschlPeakMax	0,6111	0,5843	0,5600	0,6061	0,6056	0,5932	0,6004	0,6013	0,5952
A_6	Pzvorspg1	0,8788	0,7560	0,6150	0,6273	0,5683	0,5955	0,5771	0,5553	0,6466
A_7	Pzvorspg2	0,9091	0,7651	0,6900	0,6848	0,6522	0,6386	0,6057	0,6289	0,6968
A_8	RelStellKraft	0,5556	0,3765	0,2750	0,3030	0,3602	0,3818	0,3763	0,3829	0,3764
A_9	Prutschmax	0,8434	0,5964	0,6400	0,6455	0,6398	0,6364	0,6022	0,6000	0,6504
A_{10}	Tumlauf	0,8687	0,6596	0,5600	0,5545	0,5745	0,5932	0,5538	0,5934	0,6197
A_{11}	Umotor	0,6364	0,5602	0,5700	0,5515	0,5807	0,6068	0,5538	0,5618	0,5777
A_{12}	Iunsym	0,9192	0,7500	0,6650	0,6576	0,6739	0,6455	0,6129	0,6421	0,6958

Tabelle 8.4.1: Parameter Performance - Üereinstimmung unkritischer Zustände

Tabelle 8.4.1 liefert Aufschluss über das Potenzial der Attribute in Bezug auf ihre Tauglichkeit als Analyseparameter. Die Werte der Tabelle repräsentieren die Wahrscheinlichkeit, dass die Nullhypothese (8.4.9) erfüllt wird. Bei der Auswertung fällt auf, dass sich die Werte sämtlicher Parameter, ausgenommen A_8 „RelStellKraft“ ähnlich verhalten. Die Erfolgswahrscheinlichkeit bei der Annahme von H_0 liegt bei ca. 60%. Diese Konstanz ist auf das Bilden von Partitionen während der Clusteranalyse zurückzuführen.

Analog zur Überprüfung, ob sich Variablen unkritischer Zustände gleichen, kann ein Test auf Abgrenzung der Störungen in den Messwerten durchgeführt werden. Es wird die Statistik $u_{3,i}$ herangezogen und eine Ablehnung der Nullhypothese H_0 (8.4.11) wird angestrebt. Der Hypothesentest auf Störungsdifferenziertheit gliedert sich wie folgt:

$$\begin{aligned} &\forall i \in I : \\ &H_0 : \mu_{1,i} = \mu_{2,i} \qquad H_1 : \mu_{1,i} \neq \mu_{2,i} \end{aligned} \tag{8.4.11}$$

zum Signifikanzniveau $\alpha = 0.10$.
Entscheidungsregel: Ablehnung von H_0, wenn:

$$|u_{3,i}| > \phi\left(1 - \frac{\alpha}{2}\right) = 1,64 \tag{8.4.12}$$

Tabelle 8.4.2 beschreibt den durchschnittlichen Prozentsatz der Cluster, welche die Nullhypothese ablehnen, wiederum differenziert nach dem Parameter und der Clusterzahl.

Var.	Parameter	Clusterzahl k								∅
		3	4	5	6	7	8	9	10	
A_1	Pleer	0,4848	0,5843	0,5300	0,5061	0,4876	0,4227	0,4462	0,4237	0,4857
A_2	Plauf1	0,2576	0,4940	0,4800	0,4667	0,4472	0,4136	0,4301	0,4118	0,4251
A_3	Plauf2	0,2475	0,4849	0,4950	0,4879	0,4596	0,4205	0,4516	0,4355	0,4353
A_4	Pverschl	0,4242	0,5000	0,5000	0,4909	0,4752	0,4386	0,4534	0,4421	0,4656
A_5	PverschlPeakMax	0,1869	0,4518	0,4850	0,4727	0,4379	0,4114	0,4624	0,4592	0,4209
A_6	Pzvorspg1	0,0505	0,2711	0,2950	0,3212	0,3230	0,3000	0,3369	0,3237	0,2777
A_7	Pzvorspg2	0,0556	0,2711	0,3000	0,3212	0,3261	0,3000	0,3387	0,3224	0,2794
A_8	RelStellKraft	0,0556	0,2771	0,3600	0,3909	0,3758	0,3364	0,3925	0,3882	0,3220
A_9	Prutschmax	0,4343	0,6446	0,6150	0,5818	0,5217	0,4659	0,4892	0,4605	0,5266
A_{10}	Tumlauf	0,7172	0,7590	0,7850	0,7333	0,6522	0,5182	0,5556	0,5171	0,6547
A_{11}	Umotor	0,7172	0,7560	0,7000	0,6394	0,5714	0,4841	0,5018	0,4579	0,6035
A_{12}	Iunsym	0,0909	0,3042	0,3100	0,3242	0,3323	0,3068	0,3351	0,3250	0,2911

Tabelle 8.4.2: Parameter Performance - Abgrenzung der Störungen

Eine Analyse der Werte ergibt, dass erneut der Parameter „RelStellKraft“ (A_8) schwache Ergebnisse liefert. Hinzu kommen die Variablen A_6, A_7 sowie A_{12}, welche ebenfalls durchgehend Werte unter 40% liefern.
Dieser Bewertung folgend, wird eine Untermenge der Parameter gebildet, die als Grundlage für die weitere Analyse dienen soll. Dabei wird die Anzahl der Analyseparameter auf 8 reduziert.

$$I^S = \{A_1, A_2A_3, A_4, A_5, A_9, A_{10}, A_{11}\} \tag{8.4.13}$$

Erfüllt ein Parameter beide Kriterien innerhalb eines Clusters (Ablehnung der Alternativhypothese $H1$ für den Test 8.4.9 und Ablehnung der Nullhypothese $H0$ für 8.4.11), so wird dieser für jenes Cluster als *signifikant* deklariert (Signifikanzkriterium).

Die neue Parametermenge dient nun als Grundlage für die Analyse der Clusterergebnisse. Jedes Cluster wird auf die Erfüllung des Signifikanzkriteriums geprüft, indem die Parameter, welche das Kriterium nicht einhalten verworfen werden. Als Qualitätsmaß für ein Cluster wird dann die Anzahl der Parameter herangezogen, welche die Tests „bestehen“, wobei die Objektstruktur des Clusters C_l folgende Bedingungen erfüllen muss:

$$N^S_{C_l} > 0 \qquad \text{(vgl. 8.4.1)} \tag{8.4.14}$$

$$N^{\bar{S}}_{C_l} > 0 \qquad \text{(vgl. 8.4.2)} \tag{8.4.15}$$

Tabelle 8.4.3 zeigt einen Überblick der *Clusterperformance*. Man stellt fest, dass in den meisten Fällen nur wenige (im Mittel etwa 2) Parameter das Kriterium erfüllen. Demzufolge kann nur ein Bruchteil der untersuchten Cluster insofern überzeugen, als dass mindestens 6 Parameter aus I^S die Tests „bestehen“. Für diese Menge relevanter Cluster soll nun die Objektstruktur untersucht werden.

	Clusterzahl k							
	3	4	5	6	7	8	9	10
mittlere Qualität	2,729	3,163	2,716	2,314	1,923	1,570	1,573	1,536
Anzahl analysierter Cluster	188	258	148	245	234	323	391	558
Anzahl relevanter Cluster	28	73	35	51	38	50	60	78

Tabelle 8.4.3: Cluster Qualität

Ziel der weiteren Analyse sei nun die Extraktion optimaler Objektstrukturen aus der Clustermenge, die sich durch eine ausgeprägte Abstufung der Störungen auszeichnen.
Man stellt zunächst fest, dass die Zusammensetzung der 413 Cluster aus 92 eindeutigen Objektgruppen hervorgeht. Somit stimmen die Objektstrukturen der Cluster mehrheitlich überein und nur 52 Cluster weisen eine Objektmenge auf, die sich in keinem der anderen Cluster wiederfindet.

Die gegebenen Objektstrukturen sollen nun detailliert untersucht werden. Dabei wird zunächst das Vorkommen jedes einzelnen der $M = 111$ Objekte in den relevanten Clustern gezählt und ein *Score* ermittelt, wobei ein Vorkommen stärker gewichtet wird wenn mehr als 6 bzw. 7 Parameter das Kriterium für das untersuchte Cluster erfüllen. Dabei resultiert eine Untermenge von 101 Objekten, die mindestens einem der 413 Clustern angehören und somit Teil eines oder mehreren der 92 Objektgruppen sind.
20 Objekte erreichen einen besonders hohen Score und können analog zur Relevanz von Clustern als *relevant* gekennzeichnet werden. Diese Gruppe von 20 Objekten dominieren 348 der 431 signifikanten Cluster.

Um Zusammenstellungen von Objekten, die insofern besonders gut harmonieren, als dass sie häufig im Verbund Teil von relevanten Clustern sind zu identifizieren, werden Tupel unterschiedlicher Größe gebildet, wobei gilt:

$$n^* = 20,$$
$$k^* \in [2;7];\ k \in \mathbb{N}$$

Anzahl der Kombinationen bzw. Tupel:

$$\binom{n^*}{k^*} \in \{190, 1140, 4845, 15504, 38760, 77520\} \qquad k^* \in [2;7],\ k \in \mathbb{N} \tag{8.4.16}$$

$$\sum_{k^*=2}^{7} \binom{n^*}{k^*} = 137959 \tag{8.4.17}$$

Nach dem iterativen Durchlaufen sämtlicher Tupel kommt man zu dem Ergebnis, dass für alle $k^* \in [2;7]$, $k \in \mathbb{N}$ jede mögliche Objektmenge mindestens

einem signifikanten Cluster angehört. Dies unterstreicht die oben angesprochene Dominanz der $n^* = 20$ Objekte in der Menge relevanter Cluster. Die Tabelle 8.4.4 zeigt die Resultate der Analyse.

	Tupelgröße k^*					
	2	3	4	5	6	7
Tupelanzahl	190	1140	4845	15504	38760	77520
mittleres Vorkommen in signifikaten Clustern (max. 348)	199,6	160,6	134,9	117,9	106,5	98,7
mittleres Vorkommen in relevanten Objektstrukturen (max. 92)	42,4	29,9	21,8	16,5	12,8	10,3

Tabelle 8.4.4: Vorkommen von Objekttupeln

Rückblickend auf die Ergebnisse der Clusteranalyse nach den Weichenstammdaten unter der Verwendung der Methode k-Modes, soll abschließend geprüft werden, ob sich die $n^* = 20$ relevanten Objekte mehrheitlich gruppiert auf die Cluster verteilen. Dieses Vorgehen kann als Verifizierung der Clusterzusammenführung verstanden werden.
Im Rahmen des Clusterings (Kapitel 5.4) wurden 3041 optimale Lösungen generiert. Für jede dieser Lösungen wurden die Verteilungen der relevanten Objekte sämtlicher Tupel registriert und bei Übereinstimmung der Cluster weiterverarbeitet. Ein Anteil von 77% aller Lösungen kann ein Tupel einer jeden Größe $k^* \in [2;7], k^* \in \mathbb{N}$ vorweisen. Dies bedeutet, dass ein Cluster existiert, welches mindestens 7 der relevanten Objekte enthält.
Eine Partition der Clusterzahl $k = 3$ sticht dabei heraus, indem in genau einem Cluster alle relevanten Objekte zusammengefasst werden. Im Gegensatz dazu existiert eine Lösung ($k = 10$), die sich durch weniger als 4 gruppierte relevante Objekte über alle Cluster auszeichnet.

8.5 Umsetzung in KNIME

Bekanntermaßen erfolgt die Aufzeichnung der Störungen unabhängig von der Datenquelle SIDIS W. In KNIME wird dies durch ein neues Format der Quelldaten (Microsoft Excel Spreadsheet) unterstrichen, wobei ein vorhandenes Modul das Einlesen dieses Datentyps ermöglicht (Abbildung 8.5.1).
Die Elemente der Störungsmenge müssen zunächst mit den untersuchten Objekten und dem Beobachtungszeitraum abgeglichen werden. Letztere Restriktion wird durch die Bedingung „Störungseintritt und -ende liegt zwischen dem ersten März und dem 30. Juni 2010“ und einem entsprechenden Filterungsmodul sichergestellt.

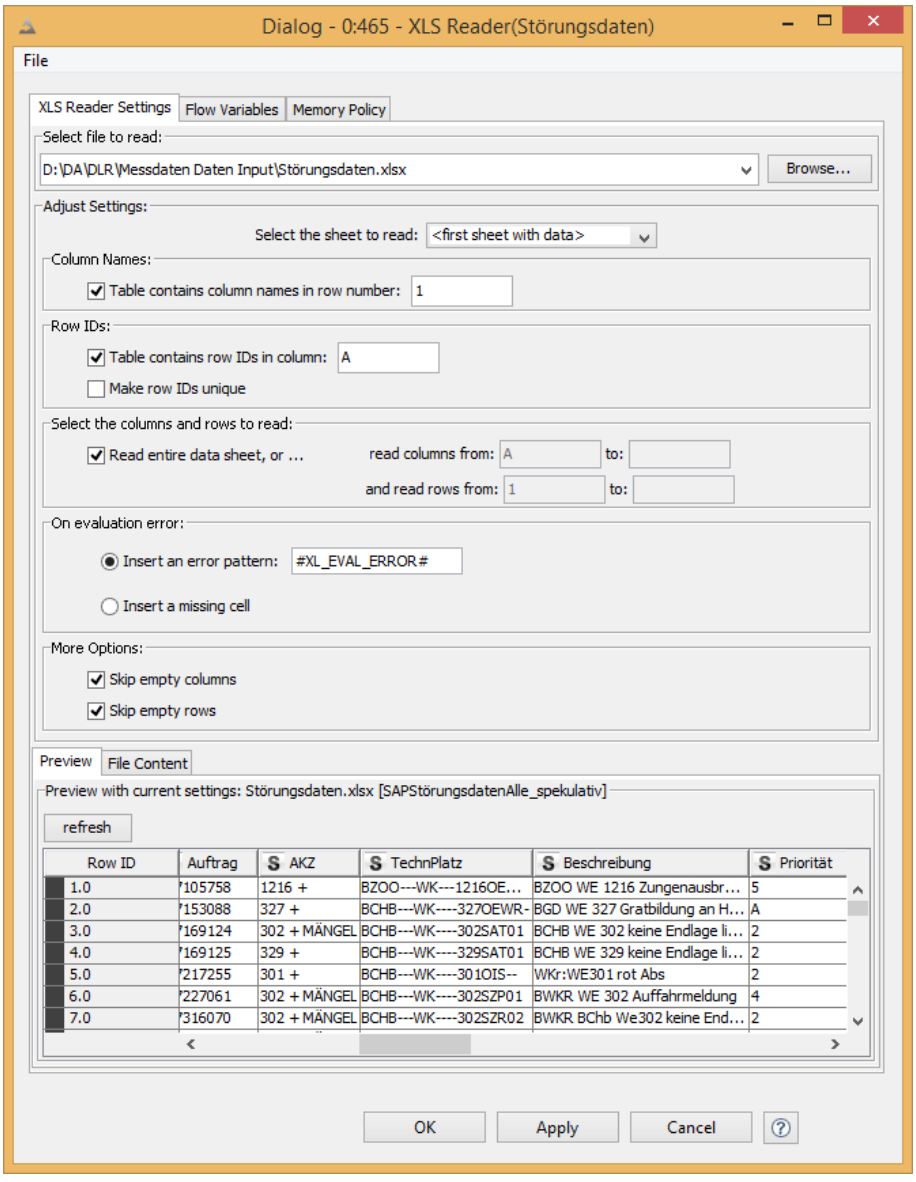

Abbildung 8.5.1: Modul zum Einlesen der Störungsdaten [KNIME]

Die Überführung der Störungen auf Objektebene wird durch zwei Tabellen abgeschlossen, welche eine Sammlung aller Störungszustände und eine Liste der Störungsobjekte mit den zugehörigen Störungseigenschaften repräsentieren. Während letztere Liste im Verlauf der weiteren Analyse zur Untersuchung der Verteilung der Prioritätsklassen und als Referenztabelle dient, komplettiert die Menge der Störungszustände unter Abgleich der Objektkennzahl die allgemeine Zustandsmenge der Messdaten.

Zu Beginn des Workflows zur Analyse der Prioritätsklassen wird die Menge der Störungsobjekte auf jene Objekte reduziert, deren zugeordneten Störungen eindeutig einer Prioritätsklasse angehören. Anschließend werden iterativ die Clustergebnisse auf Verteilung der Störungsobjekte untersucht. Der Ausschluss von Clusterergebnissen, welche nicht den Anforderungen entsprechen erfolgt erneut über einen Filterungsknoten.

Den Hauptbestandteil der operativen Analyse von Störungen auf Clusterebene bildet die Zusammenführung störungskritischer und -unkritischer Zustände. Drei parallel verlaufende Routen innerhalb einer Iteration spiegeln die Bildung der Zustandstupel pro Cluster wider (Abbildung 8.5.2). Die anschließende Verkettung erfolgt mittels zweier Join-Knoten.

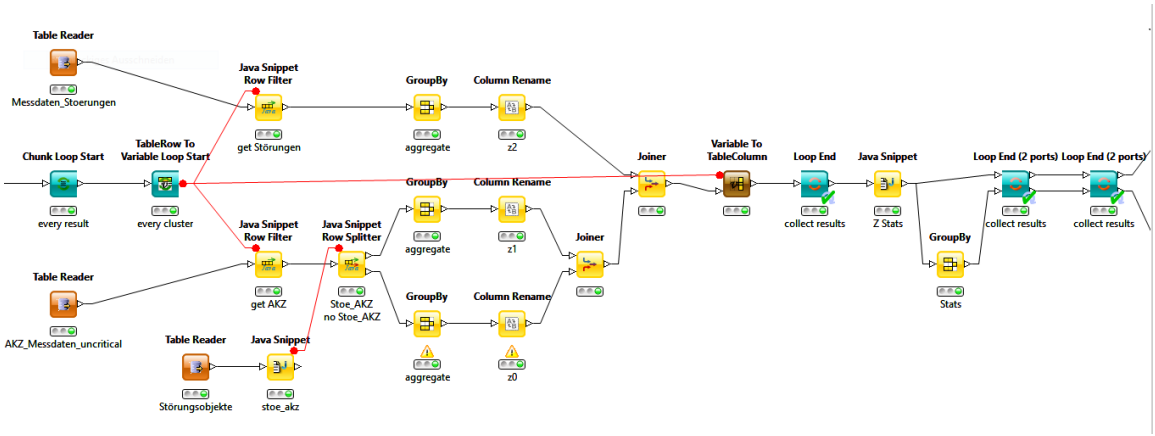

Abbildung 8.5.2: Routen der Bildung von Zustandstupeln [KNIME]

Ein Vergleich der charakterisierenden Zustände schließt sich dem an. Dabei wird der Zweistichproben-Gauß-Test innerhalb eines Java Snippet-Moduls durchgeführt (Abbildung 8.5.3) und die berechneten Werte können als zusätzliche Spalten der Clustertabelle angefügt werden.

Als Zwischenergebnis der Analyse wird eine Liste von relevanten Objekten in Form ihrer Objektkennzahl und ihres Scores gespeichert. Für diese Objektgruppe werden Untermengen unterschiedlicher Mächtigkeit gebildet, um zu prüfen, in wie fern sich diese häufig in den Objektstrukturen der Clusterergebnisse wiederfinden. Das Bilden der Tupel übernimmt dabei - analog zur Clusteranalyse (Kapitel 5.6) - der eigens erstellte Knoten namens *Column Loop Power Start* (Abbildung 8.5.4). Der Quellcode dieses Moduls wird im Anhang gelistet (A.4, A.5, A.7, A.6).

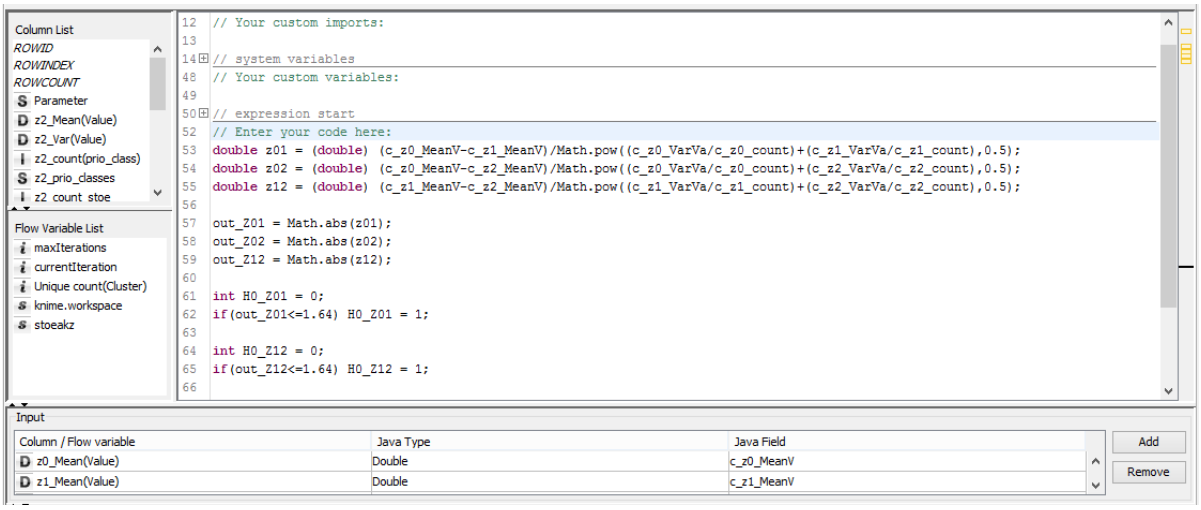

Abbildung 8.5.3: Ausschnitt des Codes zur Berechnung von Statistiken [KNIME]

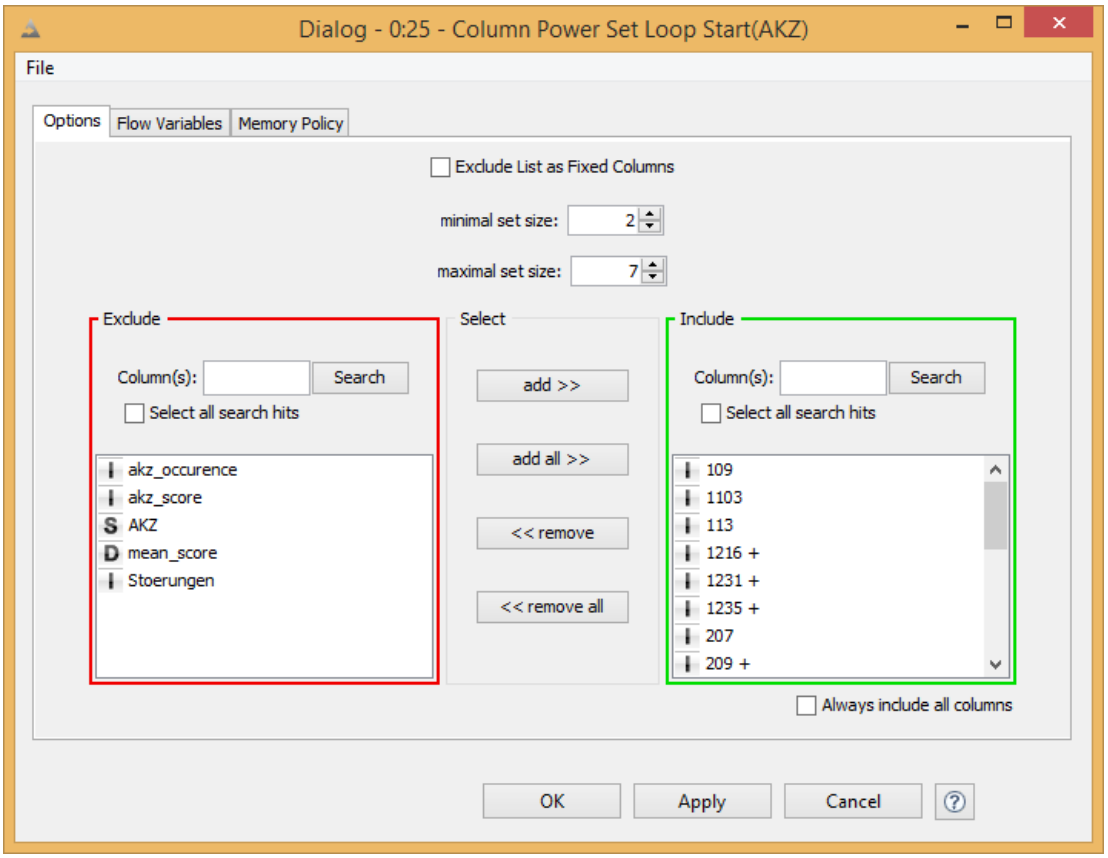

Abbildung 8.5.4: Konfiguration des Loop-Moduls zur Bildung von Untermengen [KNIME]

Kapitel 9

Schluss und Ausblick

Das Vorhaben einer durchgängigen und vollständigen Untersuchung des Verhaltens von Bahnweichen wurde umgesetzt. Dabei wurden in den Disziplinen der Cluster- und der statistischen Störungsanalyse unterschiedliche Ansätze verfolgt, um aussagekräftige Resultate zu erhalten.
Im Vorlauf der Analyse half die eingehende Erörterung der Objekte und ihrer charakterisierenden Attribute bei der Auswahl der Analyseschritte und deren Sequenz. Weiterhin war die Eingliederung der Weichenanlagen in das Feld des Plant Asset Management für das Verständnis von der Organisation der Instandhaltung förderlich.
Die anschließende Überführung von Faktoren und Rollen der Weichendiagnostik in statistische Größen stellte einen Leitfaden für den Umgang mit Weichenzuständen dar und führte den Variablenbegriff ein.

Als Hilfestellung für die Instandhaltungsplanung soll die Clusteranalyse dienen. Die festgestellte Ähnlichkeit von Objekten einer homogenen Gruppe findet Berücksichtigung in der Planung und Durchführung von Instandhaltungsmaßnahmen für ebendiese Objekte. Dabei wurde zunächst auf konventionelle Analysemethoden zurückgegriffen, um anschließend einen Vergleich mit einer modernen Disziplin des Graph Clustering anzustellen. Ausgehend von den Lösungen einer Partitionsmethode, wurden dabei in einem abstrakten Suchraum bessere Partitionen gesucht. Dies resultierte in zahlreichen Verbesserungen, wobei jedoch der Qualitätszuwachs gering ausfiel.
Mit dem Ziel einer Abbildung von Produktdaten der Weichen auf ihr technisches Verhalten, erfolgte eine Zusammenführung von Partitionen unterschiedlicher Verfahren. Dabei wurde anhand des χ^2-Kontingenzkoeffizienten ein Gütemaß eingeführt und ausführlich bewiesen.
Die Ausprägungen des Maßes dienten anschließend als Grundlage für eine Selektion der Clusterlösungen. Zum Einen führte die Filterung dabei zu einer Verkleinerung der Datenmenge, die als Input für folgende rechenintensive Operationen dient. Zum Anderen stellt sie ein analytisches Mittel dar, so dass die

übrigen Ergebnisse auf Grund nahezu übereinstimmender Objektstrukturen besonders relevant sind.

Die abschließende Störungsanalyse erlaubt ein Urteil über die vorher als relevant gekennzeichneten Objektgruppen, indem untersucht wird, ob sich auftretende Störungen ihren Merkmalen entsprechend auf die Segmente abbilden lassen.
Dabei wurden im ersten Schritt für jede Störung charakteristische Zustände ermittelt. Anschließend wurden unter Berücksichtigung des „stärksten“ Merkmals der Störungen, der Priorität Clusterlösungen ausgeschlossen, die vorwiegend ungleiche Störungen gemeinsam clustern.
Schließlich soll die Störungsanalyse die Clusterergebnisse vor dem Hintergrund der Tauglichkeit als Basis für die Instandhaltungsplanung eingehend überprüfen. Aus der großen Menge an Clusterlösungen werden solche Objektmuster extrahiert werden, deren Elemente in ihrem Verhalten ein Indiz für Störungszustände liefern. Entsprechend werden Objektgruppen und Clusterlösungen hervorgehoben, die sich durch eine ausgeprägte Relevanz bzw. Signifikanz ihrer Störungsdifferenziertheit auszeichnen.

Die praktische Umsetzung der Analyseschritte in KNIME (verwendete Version: 2.6.4) gestaltete sich sehr rechenintensiv. 27 Workflows unterschiedlichen Umfangs und Komplexität waren zur Durchführung der Analyse notwendig, wobei 113 Datentabellen erzeugt wurden und Verwendung fanden. Die Operationen wurden nahezu komplett *out of the box* von Modulen des Standardrepositories bewerkstelligt. Eine Übersicht der benötigten Erweiterungen von KNIME in Form von Plug-ins befindet sich im Anhang (A.2.1). Außerordentlich viel Rechenlaufzeit nahmen Routen in Anspruch, die sich durch Verzweigungen oder Zusammenschlüsse von Kanten hervortun. Im speziellen die Implementierung der Variable Neighborhood Search Methode des Graph Clustering stellte einen enormen Aufwand dar, was in erster Linie auf das abstrakte Gefüge der Verfahrensschritte zurückzuführen ist.

Die Resultate der Arbeit stellen zweifellos eine Hilfestellung für die Instandhaltung der DB AG dar. Dies schließt sowohl die Erkenntnisse der Cluster- als auch der Störungsanalyse ein. Jedoch erlauben sie nicht eine sofortige Abkehr weg von der fristenorientierten, hin zur zustandsorientierten Instandhaltung. Um dies zu bewerkstelligen, müsste ein Diagnosesystem in einem großflächigeren Schienennetz über eine längere Zeit zum Einsatz kommen. In diesem Zuge ist eine Überprüfung des Einflusses potenzieller (externer) Faktoren notwendig. Ebenso muss eingehend überprüft werden, ob Störungen einer Weiche in ihrem Entstehen und Auftreten unabhängig von dem Verhalten anderer Weichenanlagen sind. Daneben muss eine Verifizierung der Störungsdaten sichergestellt sein, im besten Fall einhergehend mit einer weitreichenden Ursachenforschung.

Anhang A

Anhang

A.1 Algorithmen

Variable Neighborhood Decomposition Search Pseudo Algorithmus [2]

Algorithm 1 VNDS(P)

Generiere zufällig eine Lösung x;
$x \leftarrow LPAm+(x;P)$;
$s \leftarrow 1$;
while Stopbedingung noch nicht erfüllt **do**
  Generiere ein Unterproblem S von x mit zufällig gewähltem Cluster und $s-1$ benachbarten Clustern;
  Wähle zufällig $\alpha \in$ {singleton,division,neighbor,fusion,redistribution};
  $x' \leftarrow shaking(x;\alpha;S)$;
  $x' \leftarrow LPAm+(x';S)$;
  if f(x')>f(x) **then**
    $x \leftarrow LPAm(x';P)$;
    $s \leftarrow 1$;
  else
    $s \leftarrow s+1$;
    if $s > \min\{MAX\_SIZE; \#clusters(x)\}$ **then**
      $s \leftarrow 1$;
    end if
  end if
end while
return x

A.2 KNIME

A.2.1 Abbildungen

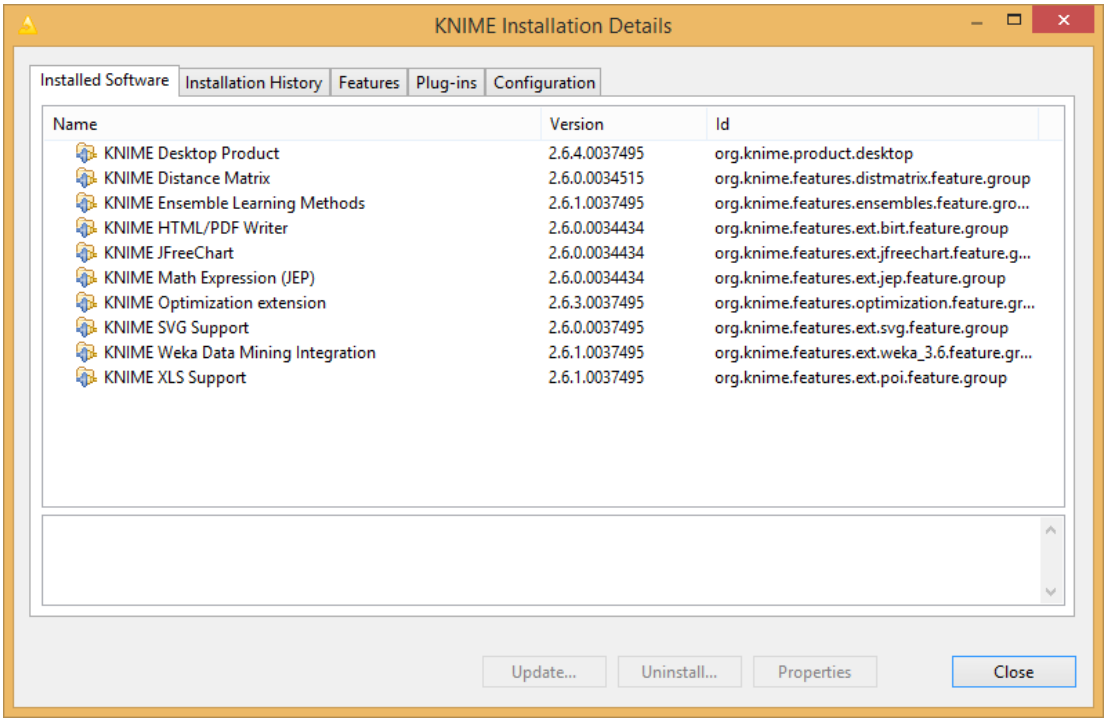

Abbildung A.2.1: Verwendete Extensions und Plug-ins in KNIME (Version 2.6.4)

A.2.2 Quellcode

Listing A.1: Operationen: Modul „Clustering Relation Scorer“

```
1  package org.dlr.hopf.relationScorerString;
2
3  import java.lang.Math;
4
5  import org.knime.core.data.DataCell;
6  import org.knime.core.data.DataRow;
7  import org.knime.core.data.RowKey;
8  import org.knime.core.data.def.DefaultRow;
9  import org.knime.core.data.def.DoubleCell;
10 import org.knime.core.data.def.IntCell;
11 import org.knime.core.data.def.StringCell;
12 import org.knime.core.node.BufferedDataContainer;
```

```
13 import org.knime.core.node.BufferedDataTable;
14
15 public class ClusterGraph {
16   /**
17    * The weight of the edges between two clusters
18    */
19   protected int edges[][];
20
21   /**
22    * The size of each cluster (of each clustering).
23    */
24   protected int clusterSize[][];
25
26   /**
27    *
28    */
29
30   protected String ClusterOne;
31   protected String ClusterTwo;
32
33   protected String[] ClusterOneSplit;
34   protected String[] ClusterTwoSplit;
35
36   protected int ClusterStringLength;
37
38   /**
39    * The number of clusters
40    */
41   protected int nrOfClusters;
42
43   protected int rowIndex = 0;
44
45
46   protected double[] qualities;
47
48   // given vars
49   BufferedDataTable table;
50   //int clusterColIdx[] = new int[2];
51   //int NumberOfClusters = 0;
52
53   /**
54    * Constructor
55    * @param table
56    * @param clusterOneIdx
57    * @param clusterTwoIdx
58    * @throws Exception
59    */
60   public ClusterGraph(BufferedDataTable table, int NoCIdx, int
       clusterOneIdx, int clusterTwoIdx, BufferedDataContainer
       container1) throws Exception {
61
62     //NumberOfClusters = NoCIdx;
63     //clusterColIdx[0] = clusterOneIdx;
64     //clusterColIdx[1] = clusterTwoIdx;
```

```
65      this.table = table;
66
67      for(DataRow row : table) {
68        nrOfClusters = ((IntCell) row.getCell(NoCIdx)).getIntValue
              ();
69
70        ClusterOne = ((StringCell) row.getCell(clusterOneIdx)).
              getStringValue();
71        ClusterTwo = ((StringCell) row.getCell(clusterTwoIdx)).
              getStringValue();
72        ClusterOneSplit = ClusterOne.split(";");
73        ClusterTwoSplit = ClusterTwo.split(";");
74        ClusterStringLength = ClusterOneSplit.length;
75
76        edges = new int[nrOfClusters][nrOfClusters];
77        clusterSize = new int[2][nrOfClusters];
78
79        for(int i=0;i<ClusterStringLength;i++) {
80          edges[Integer.parseInt(ClusterOneSplit[i])][Integer.
                parseInt(ClusterTwoSplit[i])]++;
81          clusterSize[0][Integer.parseInt(ClusterOneSplit[i])]++;
82          clusterSize[1][Integer.parseInt(ClusterTwoSplit[i])]++;
83        }
84
85        double quality = 0;
86        double sumMaxQuotient = 0;
87        for(int i=0;i<nrOfClusters;i++) {
88          double maxQuotient = 0;
89          for(int j=0;j<nrOfClusters;j++) {
90            double links = edges[i][j];
91            double links_reverse = edges[j][i];
92            double scoreP1 = 0;
93            double scoreP2 = 0;
94            double scoreP1x = 0;
95            double scoreP2x = 0;
96            if (links > 0) {
97              scoreP1 = links/(double) clusterSize[0][i];
98              scoreP2 = links/(double) clusterSize[1][j];
99              double score = scoreP1*scoreP2;
100             quality += score;
101           }
102           if (links_reverse > 0 && links_reverse>links) {
103             scoreP1x = links_reverse/(double) clusterSize[0][j];
104             scoreP2x = links_reverse/(double) clusterSize[1][i];
105           }
106           if(scoreP1>maxQuotient || scoreP2>maxQuotient ||
                  scoreP1x>maxQuotient || scoreP2x>maxQuotient) {
107             double maxQuotient1 = Math.max(scoreP1,scoreP2);
108             double maxQuotientx = Math.max(scoreP1x, scoreP2x);
109             maxQuotient = Math.max(maxQuotient1, maxQuotientx);
110           }
111         }
112         sumMaxQuotient += maxQuotient;
113       }
```

```
114         // normalize quality
115         quality = quality / (double) nrOfClusters;
116         double exponentFractionexponent = 1;
117         double exponentFraction = ((sumMaxQuotient-(double) 1)/(
                (double) nrOfClusters)*Math.pow((double) nrOfClusters,
                exponentFractionexponent);
118         double exponent = 10;
119         if(exponentFraction>0) exponent = (double) 1 /
                exponentFraction;
120         quality = Math.pow(quality, exponent);
121
122         qualities[rowIndex] = quality;
123
124         RowKey key = new RowKey("Row_" + rowIndex);
125         DataCell[] cells = new DataCell[4];
126         cells[0] = new IntCell(nrOfClusters);
127         cells[1] = new StringCell(ClusterOne);
128             cells[2] = new StringCell(ClusterOne);
129             cells[3] = new DoubleCell(quality);
130             DataRow newRow = new DefaultRow(key, cells);
131             container1.addRowToTable(newRow);
132
133             rowIndex++;
134       }
135     }
136
137 }
```

Listing A.2: Konfiguration: Modul „Clustering Relation Scorer“

```
 1 package org.dlr.hopf.relationScorerString;
 2
 3 import org.knime.core.data.IntValue;
 4 import org.knime.core.data.StringValue;
 5 import org.knime.core.node.defaultnodesettings.
       DefaultNodeSettingsPane;
 6 import org.knime.core.node.defaultnodesettings.
       DialogComponentColumnNameSelection;
 7 import org.knime.core.node.defaultnodesettings.
       SettingsModelColumnName;
 8
 9 /**
10  * <code>NodeDialog</code> for the "RelationScorer" Node.
11  *
12  *
13  * This node dialog derives from {@link DefaultNodeSettingsPane}
        which allows
14  * creation of a simple dialog with standard components. If you
       need a more
15  * complex dialog please derive directly from
16  * {@link org.knime.core.node.NodeDialogPane}.
17  *
```

```
18  * @author Thomas Hopf (DLR)
19  */
20 public class RelationScorerNodeDialog extends
       DefaultNodeSettingsPane {
21
22   /**
23    * New pane for configuring RelationScorer node dialog.
24    * This is just a suggestion to demonstrate possible default
          dialog
25    * components.
26    */
27   @SuppressWarnings("unchecked")
28         protected RelationScorerNodeDialog() {
29     super();
30
31     addDialogComponent(
32                 new DialogComponentColumnNameSelection(
33                                 new SettingsModelColumnName(
                                       RelationScorerNodeModel.
                                       CFGKEY_NoC_COLNAME, null),
34             "Number of Cluster Column:",
35             0,
36             true,
37             IntValue.class
38         ));
39
40     addDialogComponent(
41                 new DialogComponentColumnNameSelection(
42                                 new SettingsModelColumnName(
                                       RelationScorerNodeModel.
                                       CFGKEY_CLUSTER_ONE_COLNAME,
                                       null),
43             "ClusterString one:",
44             0,
45             true,
46             StringValue.class
47         ));
48
49     addDialogComponent(
50                 new DialogComponentColumnNameSelection(
51                                 new SettingsModelColumnName(
                                       RelationScorerNodeModel.
                                       CFGKEY_CLUSTER_TWO_COLNAME,
                                       null),
52             "ClusterString two:",
53             0,
54             true,
55             StringValue.class
56         ));
57
58   }
59 }
```

Listing A.3: Node Model: Modul „Clustering Relation Scorer“

```
1  package org.dlr.hopf.relationScorerString;
2
3  import java.io.File;
4  import java.io.IOException;
5
6  import org.knime.core.data.DataCell;
7  import org.knime.core.data.DataColumnSpec;
8  import org.knime.core.data.DataColumnSpecCreator;
9  import org.knime.core.data.DataRow;
10 import org.knime.core.data.DataTableSpec;
11 import org.knime.core.data.RowKey;
12 import org.knime.core.data.StringValue;
13 import org.knime.core.data.IntValue;
14 import org.knime.core.data.def.DefaultRow;
15 import org.knime.core.data.def.DoubleCell;
16 import org.knime.core.data.def.IntCell;
17 import org.knime.core.data.def.StringCell;
18 import org.knime.core.node.BufferedDataContainer;
19 import org.knime.core.node.BufferedDataTable;
20 import org.knime.core.node.CanceledExecutionException;
21 import org.knime.core.node.ExecutionContext;
22 import org.knime.core.node.ExecutionMonitor;
23 import org.knime.core.node.InvalidSettingsException;
24 import org.knime.core.node.NodeModel;
25 import org.knime.core.node.NodeSettingsRO;
26 import org.knime.core.node.NodeSettingsWO;
27 import org.knime.core.node.defaultnodesettings.
       SettingsModelColumnName;
28
29
30 /**
31  * This is the model implementation of RelationScorer.
32  *
33  *
34  * @author Thomas Hopf (DLR)
35  */
36 public class RelationScorerNodeModel extends NodeModel {
37
38   // the logger instance
39   /*private static final NodeLogger logger = NodeLogger
40     .getLogger(RelationScorerNodeModel.class);*/
41
42   /**
43    * Key used in the settings model to store the name of the
          column that contains the number of clusters
44    */
45   static final String CFGKEY_NoC_COLNAME = "NoCColName";
46   /**
47    * Variable used to retrieve the name of the column that
          contains the cluster
48    */
```

```
49    private final SettingsModelColumnName m_NoCName = new
          SettingsModelColumnName(CFGKEY_NoC_COLNAME, null);
50
51    /**
52     * Key used in the settings model to store the name of the
            column that contains the cluster
53     */
54    static final String CFGKEY_CLUSTER_ONE_COLNAME = "
          ClusterOneColName";
55    /**
56     * Variable used to retrieve the name of the column that
            contains the cluster
57     */
58    private final SettingsModelColumnName m_clusterOneColName =
          new SettingsModelColumnName(CFGKEY_CLUSTER_ONE_COLNAME,
          null);
59
60    /**
61     * Key used in the settings model to store the name of the
            column that contains the distance matrix
62     */
63    static final String CFGKEY_CLUSTER_TWO_COLNAME = "
          ClusterTwoColName";
64    /**
65     * Variable used to retrieve the name of the column that
            contains the distance matrix
66     */
67    private final SettingsModelColumnName m_clusterTwoColName =
          new SettingsModelColumnName(CFGKEY_CLUSTER_TWO_COLNAME,
          null);
68
69
70    /**
71     * The weight of the edges between two clusters
72     */
73    protected int edges[][];
74    protected double erwedges[][];
75
76    /**
77     * The size of each cluster (of each clustering).
78     */
79    protected int clusterSize[][];
80
81    /**
82     *
83     */
84    //protected TreeMap<String, Integer> dictionaryOne;
85    //protected TreeMap<String, Integer> dictionaryTwo;
86
87    protected String ClusterOne;
88    protected String ClusterTwo;
89    protected int Iteration_1_Modes;
90    protected int Iteration_2_Modes;
91    protected int Iteration_1_Means;
```

```
92    protected int Iteration_2_Means;
93
94    protected String[] ClusterOneSplit;
95    protected String[] ClusterTwoSplit;
96
97    protected int ClusterStringLength;
98
99    /**
100    * The number of clusters
101    */
102   protected int nrOfClusters;
103
104   protected int rowIndex = 0;
105
106   BufferedDataTable table;
107   //protected DataRow newRow;
108
109
110   /**
111    * Constructor for the node model.
112    */
113   protected RelationScorerNodeModel() {
114
115     // TODO one incoming port and one outgoing port is assumed
116     super(1, 1);
117   }
118
119   /**
120    * {@inheritDoc}
121    */
122   @Override
123   protected BufferedDataTable[] execute(final BufferedDataTable
          [] inData,
124     final ExecutionContext exec) throws Exception {
125
126     // TODO do something here
127     //logger.info("Node Model Stub... this is not yet
          implemented !");
128
129     // get data from input port
130     BufferedDataTable table = inData[0];
131
132     // get data from settings
133     int NoCColIdx = table.getSpec().findColumnIndex(m_NoCName.
          getColumnName());
134     if (NoCColIdx < 0) {
135     throw new InvalidSettingsException("Invalid column selected
          for 'number of clusters column'");
136     }
137     int cluster1ColIdx = table.getSpec().findColumnIndex(
          m_clusterOneColName.getColumnName());
138     if (cluster1ColIdx < 0) {
139     throw new InvalidSettingsException("Invalid column selected
          for 'cluster column 1'");
```

```
140     }
141     int cluster2ColIdx = table.getSpec().findColumnIndex(
            m_clusterTwoColName.getColumnName());
142     if (cluster2ColIdx < 0) {
143     throw new InvalidSettingsException("Invalid column selected
            for 'cluster column 2'");
144     }
145     int it1mo = table.getSpec().findColumnIndex("
            Iteration_1_Modes");
146     int it2mo = table.getSpec().findColumnIndex("
            Iteration_2_Modes");
147     int it1me = table.getSpec().findColumnIndex("
            Iteration_1_Means");
148     int it2me = table.getSpec().findColumnIndex("
            Iteration_2_Means");
149
150     DataTableSpec outputSpec1 = createOutputTabelSpecOne();
151     BufferedDataContainer container1 = exec.createDataContainer(
            outputSpec1);
152
153     container1 = fillOutTable(table, NoCColIdx, cluster1ColIdx,
            cluster2ColIdx, container1, it1mo, it2mo, it1me, it2me);
154
155     container1.close();
156
157     return new BufferedDataTable[]{container1.getTable()};
158   }
159
160   public BufferedDataContainer fillOutTable(BufferedDataTable
          table, int NoCIdx, int clusterOneIdx, int clusterTwoIdx,
          BufferedDataContainer container1, int it1mo, int it2mo, int
           it1me, int it2me) throws Exception {
161
162     //NumberOfClusters = NoCIdx;
163     //clusterColIdx[0] = clusterOneIdx;
164     //clusterColIdx[1] = clusterTwoIdx;
165     this.table = table;
166
167     for(DataRow row : table) {
168
169     nrOfClusters = ((IntCell) row.getCell(NoCIdx)).getIntValue()
            ;
170
171     Iteration_1_Modes = ((IntCell) row.getCell(it1mo)).
            getIntValue();
172     Iteration_2_Modes = ((IntCell) row.getCell(it2mo)).
            getIntValue();
173     Iteration_1_Means = ((IntCell) row.getCell(it1me)).
            getIntValue();
174     Iteration_2_Means = ((IntCell) row.getCell(it2me)).
            getIntValue();
175
176     ClusterOne = ((StringCell) row.getCell(clusterOneIdx)).
            getStringValue();
```

```
177     ClusterTwo = ((StringCell) row.getCell(clusterTwoIdx)).
            getStringValue();
178     ClusterOneSplit = ClusterOne.split(";");
179     ClusterTwoSplit = ClusterTwo.split(";");
180     ClusterStringLength = ClusterOneSplit.length;
181
182     edges = new int[nrOfClusters][nrOfClusters];
183     erwedges = new double[nrOfClusters][nrOfClusters];
184     clusterSize = new int[2][nrOfClusters];
185
186     for(int i=0;i<ClusterStringLength;i++) {
187       edges[Integer.parseInt(ClusterOneSplit[i])][Integer.
              parseInt(ClusterTwoSplit[i])]++;
188       clusterSize[0][Integer.parseInt(ClusterOneSplit[i])]++;
189       clusterSize[1][Integer.parseInt(ClusterTwoSplit[i])]++;
190     }
191
192     double relDiff = 0;
193     double chi2 = 0;
194     for(int i=0;i<nrOfClusters;i++) {
195       for(int j=0;j<nrOfClusters;j++) {
196         erwedges[i][j] = ((double)(clusterSize[0][i]*clusterSize
                [1][j]))/(double)ClusterStringLength;
197         relDiff = Math.pow((edges[i][j]-erwedges[i][j]),(double)
                2)/erwedges[i][j];
198         chi2 += relDiff;
199       }
200     }
201     double k1 = Math.pow((chi2/(chi2+ClusterStringLength)),0.5);
202     double k1max = Math.pow((((double)nrOfClusters-(double)1)/(
            double)nrOfClusters),0.5);
203     double kstern = k1/k1max;
204
205     RowKey key = new RowKey("Row_" + rowIndex);
206     DataCell[] cells = new DataCell[8];
207     cells[0] = new IntCell(nrOfClusters);
208     cells[1] = new StringCell(ClusterOne);
209     cells[2] = new StringCell(ClusterTwo);
210     cells[3] = new DoubleCell(kstern);
211     cells[4] = new IntCell(Iteration_1_Modes);
212     cells[5] = new IntCell(Iteration_2_Modes);
213     cells[6] = new IntCell(Iteration_1_Means);
214     cells[7] = new IntCell(Iteration_2_Means);
215     DataRow newRow = new DefaultRow(key, cells);
216     container1.addRowToTable(newRow);
217
218     rowIndex++;
219     }
220
221     return container1;
222   }
223
224   /**
225    * {@inheritDoc}
```

```
226     */
227    @Override
228    protected void reset() {
229      // TODO Code executed on reset.
230      // Models build during execute are cleared here.
231      // Also data handled in load/saveInternals will be erased
           here.
232    }
233
234    /**
235     * {@inheritDoc}
236     */
237    @Override
238    protected DataTableSpec[] configure(final DataTableSpec[]
         inSpecs)
239      throws InvalidSettingsException {
240
241      // if not configured yet choose string columns (prefer
           columns beginning with 'cluster')
242      String NoCCol = m_NoCName.getColumnName();
243      String colName1 = m_clusterOneColName.getColumnName();
244      String colName2 = m_clusterTwoColName.getColumnName();
245
246      // configure first cluster column name
247      if (NoCCol == null) {
248        for (int i = 0; i < inSpecs[0].getNumColumns(); i++) {
249      DataColumnSpec colSpec = inSpecs[0].getColumnSpec(i);
250      if (colSpec.getType().isCompatible(IntValue.class)) {
251        String colname = colSpec.getName();
252        if ((colName1 == null || !colName1.equals(colname)) && (
             colName2 == null || !colName2.equals(colname))) {
253        NoCCol = colname;
254        }
255      }
256      }
257        if (colName1 == null) {
258          throw new InvalidSettingsException("Input table must at
               least contain two string column.");
259        }
260        m_NoCName.setStringValue(NoCCol);
261      }
262
263      // configure first cluster column name
264      if (colName1 == null) {
265        for (int i = 0; i < inSpecs[0].getNumColumns(); i++) {
266      DataColumnSpec colSpec = inSpecs[0].getColumnSpec(i);
267      if (colSpec.getType().isCompatible(StringValue.class)) {
268        String colname = colSpec.getName();
269        if (colName2 == null || !colName2.equals(colname)) {
270        if (colName1 == null || (!colName1.startsWith("Cluster")
             && colname.startsWith("Cluster"))) {
271          colName1 = colname;
272        }
273        }
```

```
274     }
275       }
276       if (colName1 == null) {
277         throw new InvalidSettingsException("Input table must at
              least contain two string column.");
278       }
279       m_clusterOneColName.setStringValue(colName1);
280     }
281
282     // configure second cluster column name
283     if (colName2 == null) {
284       for (int i = inSpecs[0].getNumColumns()-1; i >= 0 ; i--) {
285     DataColumnSpec colSpec = inSpecs[0].getColumnSpec(i);
286     if (colSpec.getType().isCompatible(StringValue.class)) {
287       String colname = colSpec.getName();
288       if (!colName1.equals(colname)) {
289       if (colName2 == null || (!colName2.startsWith("Cluster")
              && colname.startsWith("Cluster"))) {
290         colName2 = colname;
291       }
292       }
293     }
294       }
295       if (colName2 == null) {
296         throw new InvalidSettingsException("Input table must at
              least contain two string column.");
297       }
298       m_clusterTwoColName.setStringValue(colName2);
299     }
300
301     // check if the same column was selected two times
302     if (colName1.equals(colName2)) {
303       throw new InvalidSettingsException("You need to choose two
            different columns for the cluster assignments.");
304     }
305
306     // create output specs
307     return new DataTableSpec[]{createOutputTabelSpecOne()};
308   }
309
310   /**
311    * Creates the specification for the new generated table (
          output port 1)
312    * @return
313    * @throws InvalidSettingsException
314    */
315   protected DataTableSpec createOutputTabelSpecOne() throws
        InvalidSettingsException {
316     // the data table spec of the single output table
317     DataColumnSpec[] allColSpecs = new DataColumnSpec[8];
318     allColSpecs[0] =
319     new DataColumnSpecCreator(m_NoCName.getColumnName(), IntCell
          .TYPE).createSpec();
320     allColSpecs[1] =
```

```
321       new DataColumnSpecCreator(m_clusterOneColName.getColumnName
              (), StringCell.TYPE).createSpec();
322       allColSpecs[2] =
323         new DataColumnSpecCreator(m_clusterTwoColName.
              getColumnName(), StringCell.TYPE).createSpec();
324       allColSpecs[3] =
325       new DataColumnSpecCreator("Quality Measure", DoubleCell.TYPE
              ).createSpec();
326       allColSpecs[4] =
327       new DataColumnSpecCreator("Iteration_1_Modes", IntCell.TYPE)
              .createSpec();
328       allColSpecs[5] =
329       new DataColumnSpecCreator("Iteration_2_Modes", IntCell.TYPE)
              .createSpec();
330       allColSpecs[6] =
331       new DataColumnSpecCreator("Iteration_1_Means", IntCell.TYPE)
              .createSpec();
332       allColSpecs[7] =
333       new DataColumnSpecCreator("Iteration_2_Means", IntCell.TYPE)
              .createSpec();
334       return new DataTableSpec(allColSpecs);
335     }
336
337     /**
338      * {@inheritDoc}
339      */
340     @Override
341     protected void saveSettingsTo(final NodeSettingsWO settings) {
342
343     // TODO save user settings to the config object.
344
345       m_NoCName.saveSettingsTo(settings);
346       m_clusterOneColName.saveSettingsTo(settings);
347       m_clusterTwoColName.saveSettingsTo(settings);
348     }
349
350     /**
351      * {@inheritDoc}
352      */
353     @Override
354     protected void loadValidatedSettingsFrom(final NodeSettingsRO
              settings)
355       throws InvalidSettingsException {
356
357       // TODO load (valid) settings from the config object.
358       // It can be safely assumed that the settings are valided by
              the
359       // method below.
360
361       m_NoCName.loadSettingsFrom(settings);
362       m_clusterOneColName.loadSettingsFrom(settings);
363       m_clusterTwoColName.loadSettingsFrom(settings);
364
365     }
```

```
366
367     /**
368      * {@inheritDoc}
369      */
370     @Override
371     protected void validateSettings(final NodeSettingsRO settings)
372       throws InvalidSettingsException {
373
374     // TODO check if the settings could be applied to our model
375     // e.g. if the count is in a certain range (which is ensured
           by the
376     // SettingsModel).
377     // Do not actually set any values of any member variables.
378
379       m_NoCName.validateSettings(settings);
380       m_clusterOneColName.validateSettings(settings);
381       m_clusterTwoColName.validateSettings(settings);
382     }
383
384     /**
385      * {@inheritDoc}
386      */
387     @Override
388     protected void loadInternals(final File internDir,
389       final ExecutionMonitor exec) throws IOException,
390       CanceledExecutionException {
391
392     // TODO load internal data.
393     // Everything handed to output ports is loaded automatically (
           data
394     // returned by the execute method, models loaded in
           loadModelContent,
395     // and user settings set through loadSettingsFrom - is all
           taken care
396     // of). Load here only the other internals that need to be
           restored
397     // (e.g. data used by the views).
398
399     }
400
401     /**
402      * {@inheritDoc}
403      */
404     @Override
405     protected void saveInternals(final File internDir,
406       final ExecutionMonitor exec) throws IOException,
407       CanceledExecutionException {
408
409     // TODO save internal models.
410     // Everything written to output ports is saved automatically (
           data
411     // returned by the execute method, models saved in the
           saveModelContent,
```

```
412     // and user settings saved through saveSettingsTo - is all
            taken care
413     // of). Save here only the other internals that need to be
            preserved
414     // (e.g. data used by the views).
415
416     }
417
418 }
```

Listing A.4: Operationen: Modul „Column Power Set Loop Start“

```
 1 package org.dlr.nywelt.columnPowerSetLoopStart;
 2
 3 import org.knime.core.data.DataValue;
 4 import org.knime.core.node.defaultnodesettings.
       DefaultNodeSettingsPane;
 5 import org.knime.core.node.defaultnodesettings.
       DialogComponentColumnFilter;
 6 import org.knime.core.node.defaultnodesettings.
       DialogComponentNumber;
 7 import org.knime.core.node.defaultnodesettings.
       DialogComponentBoolean;
 8 import org.knime.core.node.defaultnodesettings.
       SettingsModelFilterString;
 9 import org.knime.core.node.defaultnodesettings.
       SettingsModelIntegerBounded;
10 import org.knime.core.node.defaultnodesettings.
       SettingsModelBoolean;
11
12 /**
13  * <code>NodeDialog</code> for the "ColumnPermLoopStart" Node.
14  *
15  *
16  * This node dialog derives from {@link DefaultNodeSettingsPane}
        which allows
17  * creation of a simple dialog with standard components. If you
        need a more
18  * complex dialog please derive directly from
19  * {@link org.knime.core.node.NodeDialogPane}.
20  *
21  * @author Christian Nywelt, Thomas Boehm, Thomas Hopf (DLR
        Braunschweig)
22  */
23 public class ColumnPowerSetLoopStartNodeDialog extends
       DefaultNodeSettingsPane {
24
25   /**
26    * Creates a new dialog.
27    */
28   @SuppressWarnings("unchecked")
29   public ColumnPowerSetLoopStartNodeDialog() {
```

```
30    super();
31
32    addDialogComponent(new DialogComponentBoolean(
33      new SettingsModelBoolean(
34        ColumnPowerSetLoopStartNodeModel.EXCL_AS_FIXED, false), "
              Exclude List as Fixed Columns"));
35
36    addDialogComponent(new DialogComponentNumber(
37      new SettingsModelIntegerBounded(
38        ColumnPowerSetLoopStartNodeModel.MIN_SET_SIZE,
39        ColumnPowerSetLoopStartNodeModel.INITIAL_MIN_SET_SIZE,
40        0, Integer.MAX_VALUE),
41        "minimal set size:", /*step*/ 1, /*componentwidth*/ 5));
42
43    addDialogComponent(new DialogComponentNumber(
44      new SettingsModelIntegerBounded(
45        ColumnPowerSetLoopStartNodeModel.MAX_SET_SIZE,
46        ColumnPowerSetLoopStartNodeModel.INITIAL_MAX_SET_SIZE,
47        0, Integer.MAX_VALUE),
48        "maximal set size:", /*step*/ 1, /*componentwidth*/ 5));
49
50    addDialogComponent(new DialogComponentColumnFilter(
51      new SettingsModelFilterString(
            ColumnPowerSetLoopStartNodeModel.CFG_COLUMNS),
52        0, true, DataValue.class));
53    }
54  }
```

Listing A.5: Operationen: Modul „Column Power Set Loop Start“(2)

```
1  package org.dlr.nywelt.columnPowerSetLoopStart;
2
3  import java.util.LinkedList;
4  import java.util.List;
5  import java.util.NoSuchElementException;
6
7  /**
8   * Calculates the power set of a given set.
9   * @author Christian Nywelt, Thomas Hopf
10  */
11 public class PowerSet {
12   String values[];
13   int indicator[];
14   int minSize;
15   int unmodifiedMinSize; // because is used for size calculation
          and minSize gets changed when the algorithm is executed
16   int maxSize;
17   boolean hasNext = true;
18   /**
19    * maximum number of returned elements
20    */
21   protected int powerSetSize = -1;
```

```
22
23    /**
24     *
25     * @param values
26     * @param minSize
27     * @param maxSize
28     */
29    public PowerSet(List<String> values, int minSize, int maxSize)
          {
30      String[] arr = new String[values.size()];
31      values.toArray(arr);
32      initVars(arr, minSize, maxSize);
33    }
34
35    /**
36     * Constructor
37     * @param values
38     * @param minSize
39     * @param maxSize
40     */
41    public PowerSet(String values[], int minSize, int maxSize) {
42      initVars(values, minSize, maxSize);
43    }
44
45
46    /**
47     *
48     * @param values
49     * @param minSize
50     * @param maxSize
51     */
52    protected void initVars(String values[], int minSize, int
          maxSize) {
53      this.values = values;
54      this.minSize = unmodifiedMinSize = minSize;
55      if(maxSize<0) {
56        this.maxSize = values.length;
57      } else {
58        this.maxSize = (maxSize == 0) ? values.length : maxSize;
59      }
60      init(minSize);
61    }
62
63    /**
64     * Calculates the number of sets in the power set
65     * @return
66     */
67    protected int calculateSize(int elementCount, int minSize, int
          maxSize) {
68      int counter = 0;
69      // calculate the number of all sets
70      for (int i = minSize; i <= maxSize; i++) {
71        counter += nCr(elementCount, i);
72      }
```

```
73      return counter;
74    }
75
76
77    /**
78     * Returns the number of sets in the power set.
79     * @return
80     */
81    public int getSize() {
82      if (powerSetSize == -1) powerSetSize = calculateSize(values.
            length, unmodifiedMinSize, maxSize);
83      return powerSetSize;
84    }
85
86
87    /**
88     * Calculates the binomial coefficient
89     * @param n
90     * @param k
91     * @return
92     */
93    protected static int nCr(int n, int k) {
94      if (k == 0 || k == n) return 1;
95      else if (2*k > n) return nCr(n, n-k);
96      else {
97        double res = 1;
98        for(double i = 1; i <= k; i++) {
99          res *= (n-i+1)/i;
100       }
101       return (int) res;
102     }
103   }
104
105   /**
106    * init with exact amount of contains
107    */
108   protected void init(int n) {
109     indicator = new int[n];
110     for(int i = 0; i < n; i++) {
111       indicator[i] = i;
112     }
113   }
114
115   /**
116    * generate next set
117    * @return true if there is a next set
118    */
119   protected boolean generateNextSet() {
120     int max = values.length-1;
121     int i = indicator.length-1;
122
123     // serach for the first value that can be increased
124     while (!(indicator[i] < max)) {
125       i--;
```

```
126         max--;
127         if (i < 0) return false;
128       }
129
130       // increase that value and reinit the following ones
131       int value = ++indicator[i];
132       while (++i < indicator.length) {
133         indicator[i] = ++value;
134       }
135
136       return true;
137     }
138
139
140     /**
141      * generates a set from the current
142      */
143     protected List<String> getCurrentSet() {
144       LinkedList<String> subSet = new LinkedList<String>();
145       for(int i : indicator) subSet.add(values[i]);
146       return subSet;
147     }
148
149     /**
150      *
151      * @return
152      */
153     public boolean hasNext() {
154       // TODO Auto-generated method stub
155       return hasNext;
156     }
157
158     /**
159      *
160      * @return
161      */
162     public List<String> next() {
163       if (!hasNext()) throw new NoSuchElementException();
164
165       // create current set
166       List<String> res = getCurrentSet();
167
168       // generate next set
169       if (minSize > 0) {
170         hasNext = generateNextSet();
171       } else {
172         hasNext = false;
173       }
174
175       if (!hasNext && minSize < maxSize) {
176         minSize++;
177         init(minSize);
178         hasNext = true;
179       }
```

```
180
181     return res;
182   }
183 }
```

Listing A.6: Konfiguration: Modul „Column Power Set Loop Start“

```
 1 package org.dlr.nywelt.columnPowerSetLoopStart;
 2
 3 import org.knime.core.data.DataValue;
 4 import org.knime.core.node.defaultnodesettings.
       DefaultNodeSettingsPane;
 5 import org.knime.core.node.defaultnodesettings.
       DialogComponentColumnFilter;
 6 import org.knime.core.node.defaultnodesettings.
       DialogComponentNumber;
 7 import org.knime.core.node.defaultnodesettings.
       DialogComponentBoolean;
 8 import org.knime.core.node.defaultnodesettings.
       SettingsModelFilterString;
 9 import org.knime.core.node.defaultnodesettings.
       SettingsModelIntegerBounded;
10 import org.knime.core.node.defaultnodesettings.
       SettingsModelBoolean;
11
12 /**
13  * <code>NodeDialog</code> for the "ColumnPermLoopStart" Node.
14  *
15  *
16  * This node dialog derives from {@link DefaultNodeSettingsPane}
        which allows
17  * creation of a simple dialog with standard components. If you
       need a more
18  * complex dialog please derive directly from
19  * {@link org.knime.core.node.NodeDialogPane}.
20  *
21  * @author Christian Nywelt, Thomas Boehm, Thomas Hopf (DLR
       Braunschweig)
22  */
23 public class ColumnPowerSetLoopStartNodeDialog extends
       DefaultNodeSettingsPane {
24
25   /**
26    * Creates a new dialog.
27    */
28   @SuppressWarnings("unchecked")
29   public ColumnPowerSetLoopStartNodeDialog() {
30     super();
31
32     addDialogComponent(new DialogComponentBoolean(
33         new SettingsModelBoolean(
```

```
34          ColumnPowerSetLoopStartNodeModel.EXCL_AS_FIXED, false),
                "Exclude List as Fixed Columns"));
35
36      addDialogComponent(new DialogComponentNumber(
37          new SettingsModelIntegerBounded(
38              ColumnPowerSetLoopStartNodeModel.MIN_SET_SIZE,
39              ColumnPowerSetLoopStartNodeModel.
                    INITIAL_MIN_SET_SIZE,
40              0, Integer.MAX_VALUE),
41              "minimal set size:", /*step*/ 1, /*componentwidth*/
                    5));
42
43      addDialogComponent(new DialogComponentNumber(
44          new SettingsModelIntegerBounded(
45              ColumnPowerSetLoopStartNodeModel.MAX_SET_SIZE,
46              ColumnPowerSetLoopStartNodeModel.
                    INITIAL_MAX_SET_SIZE,
47              0, Integer.MAX_VALUE),
48              "maximal set size:", /*step*/ 1, /*componentwidth*/
                    5));
49
50      addDialogComponent(new DialogComponentColumnFilter(
51          new SettingsModelFilterString(
                ColumnPowerSetLoopStartNodeModel.CFG_COLUMNS),
52          0, true, DataValue.class));
53    }
54 }
```

Listing A.7: Node Model: Modul „Column Power Set Loop Start“

```
1  package org.dlr.nywelt.columnPowerSetLoopStart;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.text.NumberFormat;
6  import java.util.LinkedList;
7  import java.util.List;
8
9  import org.knime.core.data.DataCell;
10 import org.knime.core.data.DataColumnSpec;
11 import org.knime.core.data.DataColumnSpecCreator;
12 import org.knime.core.data.DataRow;
13 import org.knime.core.data.DataTableSpec;
14 import org.knime.core.data.RowKey;
15 import org.knime.core.data.container.ColumnRearranger;
16 import org.knime.core.data.def.DefaultRow;
17 import org.knime.core.data.def.StringCell;
18 import org.knime.core.node.BufferedDataContainer;
19 import org.knime.core.node.BufferedDataTable;
20 import org.knime.core.node.CanceledExecutionException;
21 import org.knime.core.node.ExecutionContext;
22 import org.knime.core.node.ExecutionMonitor;
```

```
23 import org.knime.core.node.InvalidSettingsException;
24 import org.knime.core.node.NodeModel;
25 import org.knime.core.node.NodeSettingsRO;
26 import org.knime.core.node.NodeSettingsWO;
27 import org.knime.core.node.defaultnodesettings.
       SettingsModelBoolean;
28 import org.knime.core.node.defaultnodesettings.
       SettingsModelFilterString;
29 import org.knime.core.node.defaultnodesettings.
       SettingsModelIntegerBounded;
30 import org.knime.core.node.workflow.LoopStartNodeTerminator;
31
32
33 /**
34  * This is the model implementation of ColumnPermLoopStart.
35  *
36  *
37  * @author Christian Nywelt, Thomas Hopf (DLR Braunschweig)
38  */
39 public class ColumnPowerSetLoopStartNodeModel extends NodeModel
       implements LoopStartNodeTerminator {
40   //private static final NodeLogger logger = NodeLogger.
         getLogger(ColumnPowerSetLoopStartNodeModel.class);
41
42   protected static final NumberFormat numberFormat =
         NumberFormat.getIntegerInstance();
43
44   /** Constant for the initial * used in the dialog. */
45     public static final int INITIAL_MIN_SET_SIZE = 0;
46     public static final int INITIAL_MAX_SET_SIZE = 0;
47
48     /** Config keys for the settings. */
49     public static final String EXCL_AS_FIXED = "ExclAsFixed";
50     public static final String MIN_SET_SIZE = "minSetSize";
51     public static final String MAX_SET_SIZE = "maxSetSize";
52
53     /** names of the variables pushed to stack **/
54     protected static final String ITERATION_VARIABLE_NAME = "
           currentPowerSetIteration";
55     protected static final String MAX_ITERATION_VARIABLE_NAME =
           "maxPowerSetIterations";
56
57     /*
58      * Use settings models and dialog components.
59      */
60     private final SettingsModelBoolean m_exclAsfixed
61     = new SettingsModelBoolean(EXCL_AS_FIXED, true);
62
63     private final SettingsModelIntegerBounded m_minSetSize
64     = new SettingsModelIntegerBounded(MIN_SET_SIZE,
65     INITIAL_MIN_SET_SIZE, 0, Integer.MAX_VALUE);
66
67     private final SettingsModelIntegerBounded m_maxSetSize
68     = new SettingsModelIntegerBounded(MAX_SET_SIZE,
```

```
69      INITIAL_MAX_SET_SIZE, 0, Integer.MAX_VALUE);
70
71
72    /** Config key for the used columns. */
73      public static final String CFG_COLUMNS = "cfgColmns";
74
75      SettingsModelFilterString m_usedColumns = new
            SettingsModelFilterString(
            ColumnPowerSetLoopStartNodeModel.CFG_COLUMNS);
76
77
78    private boolean m_lastIteration;
79
80    private PowerSet powerSetGenerator;
81
82    private LinkedList<String> currentIncludedColumns = new
          LinkedList<String>();
83
84    private int m_iteration = 0;
85    private int m_powerSetSize = 0;
86
87    /**
88     * Creates a new model with one data out- and inport,
            respectively.
89     */
90    public ColumnPowerSetLoopStartNodeModel() {
91        super(1, 2);
92    }
93
94    /**
95     * {@inheritDoc}
96     */
97    @Override
98    protected DataTableSpec[] configure(final DataTableSpec[]
          inSpecs) throws InvalidSettingsException {
99      // TODO: check if not configured yet and select all string
            compatible columns
100
101     // check for errors
102       if ((m_usedColumns.getIncludeList().isEmpty())
103           && !m_usedColumns.isKeepAllSelected()) {
104       throw new InvalidSettingsException(
105           "No columns to iterate over selected");
106       }
107
108     if (!m_usedColumns.isKeepAllSelected()) {
109     for (String col : m_usedColumns.getIncludeList()) {
110         if (!inSpecs[0].containsName(col)) {
111         throw new IllegalArgumentException("Column '" + col
112             + "' does not exist in input table");
113           }
114       }
115       }
116
```

```
117     // check if lower bound is smaller than upper bound
118     if (m_maxSetSize.getIntValue() > 0 && (m_minSetSize.
            getIntValue() > m_maxSetSize.getIntValue())) {
119       throw new InvalidSettingsException("Minimal set size can't
              be greater than the maximal set size");
120     }
121
122     // check if ExcludedColumns is not empty when "ExclAsFixed"=
            true
123     if(m_exclAsfixed.getBooleanValue() && m_usedColumns.
            getExcludeList().isEmpty()) {
124       throw new InvalidSettingsException("Assign columns, which
              have to be 'fixed' to the Excluded List");
125     }
126
127     // check if lower bound is greater than number of columns
128     if(!m_exclAsfixed.getBooleanValue() && m_minSetSize.
            getIntValue() > m_usedColumns.getIncludeList().size()) {
129       throw new InvalidSettingsException("Minimal set size can't
              be greater than the number of included colums");
130     } else if(m_exclAsfixed.getBooleanValue() && m_minSetSize.
            getIntValue() > (m_usedColumns.getIncludeList().size() +
            m_usedColumns.getExcludeList().size())) {
131       throw new InvalidSettingsException("Minimal set size can't
              be greater than the number of all colums");
132     }
133
134     // check if maximal set size exceeds the number of columns
135     if(m_exclAsfixed.getBooleanValue() && (m_maxSetSize.
            getIntValue()>(m_usedColumns.getIncludeList().size()+
            m_usedColumns.getExcludeList().size()))) {
136       throw new InvalidSettingsException("Maximal set size can't
              be greater than the number of all columns");
137     } else if(!m_exclAsfixed.getBooleanValue() && (m_maxSetSize.
            getIntValue()>m_usedColumns.getIncludeList().size())) {
138       throw new InvalidSettingsException("Maximal set size can't
              be greater than the number of included columns");
139     }
140
141     // check if lower bound is greater than fixed columns
142     if(m_exclAsfixed.getBooleanValue() && m_usedColumns.
            getExcludeList().size()>2 && m_minSetSize.getIntValue()<
            m_usedColumns.getExcludeList().size()) {
143       throw new InvalidSettingsException("Minimal set size can't
              be less than the number of the fixed columns");
144     }
145     // check if upper bound is greater than fixed columns
146     if(m_exclAsfixed.getBooleanValue() && m_usedColumns.
            getExcludeList().size()>2 && m_maxSetSize.getIntValue()>0
              && m_maxSetSize.getIntValue()<m_usedColumns.
            getExcludeList().size()) {
147       throw new InvalidSettingsException("Maximal set size can't
              be less than the number of the fixed columns");
148     }
```

```
149
150     // create power set
151     LinkedList<String> columnNames;
152     if (m_usedColumns.isKeepAllSelected()) {
153       columnNames = new LinkedList<String>();
154       for (DataColumnSpec spec : inSpecs[0]) {
155     columnNames.add(spec.getName());
156       }
157     } else {
158       columnNames = new LinkedList<String>(m_usedColumns.
              getIncludeList());
159     }
160
161     int minSetSize = m_minSetSize.getIntValue();
162     int maxSetSize = m_maxSetSize.getIntValue();
163     /*if (minSetSize == 0 && m_usedColumns.getExcludeList().
          isEmpty()) {
164       minSetSize = 1; // remove empty element (always at first
              position)
165     }*/
166     if (minSetSize == 0) {
167       minSetSize = 1; // remove empty element (always at first
              position)
168     }
169     if(m_exclAsfixed.getBooleanValue()) {
170       minSetSize = minSetSize - m_usedColumns.getExcludeList().
              size();
171       if(maxSetSize>0) maxSetSize = maxSetSize - m_usedColumns.
              getExcludeList().size();
172       if(m_usedColumns.getExcludeList().size()>0 && maxSetSize
              ==0) maxSetSize = -1 * m_usedColumns.getExcludeList().
              size();
173     }
174
175     powerSetGenerator = new PowerSet(columnNames, minSetSize,
          maxSetSize);
176
177     m_powerSetSize = powerSetGenerator.getSize();
178     // output warning
179     setWarningMessage("Predicted number of loops: " +
          numberFormat.format(m_powerSetSize));
180     if(m_exclAsfixed.getBooleanValue()) {
181       setWarningMessage("Notice: 'Excluded' columns are instead
              FIXED. No columns excluded!");
182     }
183
184     // create output specs
185       ColumnRearranger crea;
186     try {
187       LinkedList<String> firstPowerSet = new LinkedList<String
              >();
188       // add columns in the given order (so we do not have to
              wait for the
189       for (int i = 0; i < minSetSize; i++) {
```

```
190       firstPowerSet.add(columnNames.get(i));
191         }
192         // create rearanger
193         crea = createRearranger(inSpecs[0], firstPowerSet);
194       } catch (Exception e) {
195         throw new InvalidSettingsException(e.getMessage());
196       }
197
198       // put variables on stack
199       pushFlowVariableInt(ITERATION_VARIABLE_NAME, m_iteration);
200       pushFlowVariableInt(MAX_ITERATION_VARIABLE_NAME,
              m_powerSetSize);
201
202         return new DataTableSpec[]{crea.createSpec(),
              createOutputTabelSpecTwo()};
203     }
204
205
206     /**
207      *
208      * @param inSpec
209      * @return
210      * @throws Exception
211      */
212     private ColumnRearranger createRearranger(final DataTableSpec
          inSpec, List<String> currentPowerSet) throws Exception {
213       currentIncludedColumns.clear();
214
215       // if needed assign the columns, that should be "fixed"
              included to the "ExcludeList"
216         // the ExcludedList would then have no other function
217       // add cols that should always be included
218         if (m_exclAsfixed.getBooleanValue() && !m_usedColumns.
              isKeepAllSelected() && !m_usedColumns.getExcludeList().
              isEmpty()) {
219       currentIncludedColumns.addAll(m_usedColumns.getExcludeList()
            );
220         }
221
222         currentIncludedColumns.addAll(currentPowerSet);
223
224         // create rearranger
225         String[] columnNames = new String[currentIncludedColumns.
              size()];
226         currentIncludedColumns.toArray(columnNames);
227
228         ColumnRearranger crea = new ColumnRearranger(inSpec);
229         crea.keepOnly(columnNames);
230
231         return crea;
232     }
233
234
235     /**
```

```
236     * {@inheritDoc}
237     */
238    @Override
239    protected BufferedDataTable[] execute(final BufferedDataTable
           [] inData, final ExecutionContext exec) throws Exception {
240      // put variables on stack
241      m_iteration++;
242      pushFlowVariableInt(ITERATION_VARIABLE_NAME, m_iteration);
243      pushFlowVariableInt(MAX_ITERATION_VARIABLE_NAME,
             m_powerSetSize);
244
245      // calculate next set
246      List<String> currentPowerSet = powerSetGenerator.next();
247      ColumnRearranger crea = createRearranger(inData[0].
             getDataTableSpec(), currentPowerSet);
248
249        // check if last iteration is reached
250        m_lastIteration = !powerSetGenerator.hasNext();
251
252        // create current specs for table one
253        BufferedDataTable outputTable = exec.
               createColumnRearrangeTable(inData[0], crea, exec);
254
255        // create second output table
256        BufferedDataContainer container2 = createTableTwo(inData
               [0].getSpec(), crea, exec);
257
258        return new BufferedDataTable[]{outputTable, container2.
               getTable()};
259    }
260
261
262    /**
263     *
264     * @param inputSpecs
265     * @param exec
266     * @return
267     */
268    @SuppressWarnings("static-access")
269    protected BufferedDataContainer createTableTwo(final
           DataTableSpec inputSpecs, ColumnRearranger crea, final
           ExecutionContext exec) {
270      DataTableSpec outputSpec = createOutputTabelSpecTwo();
271      BufferedDataContainer container = exec.createDataContainer(
           outputSpec);
272
273      DataTableSpec outPutSpec = crea.createSpec();
274
275      // get included and excluded columns
276      LinkedList<String> includedList = new LinkedList<String>();
277      LinkedList<String> exludedList = new LinkedList<String>();
278      for(DataColumnSpec spec : inputSpecs) {
279        String colName = spec.getName();
280        if (outPutSpec.containsName(colName)) {
```

```
281         includedList.add(colName);
282       } else {
283         exludedList.add(colName);
284       }
285     }
286 
287     // fill table
288     /*int tableLength = 0;
289     if(!m_exclAsfixed.getBooleanValue()) {
290       tableLength = Math.max(includedList.size(), exludedList.
              size());
291     } else {
292       tableLength = includedList.size() + exludedList.size();
293     }*/
294     int tableLength = Math.max(includedList.size(), exludedList.
          size());
295     for(int i = 0; i < tableLength; i++) {
296       RowKey key = new RowKey("Row" + i);
297       DataCell[] cells = new DataCell[2];
298 
299       if (i < includedList.size()) {
300         cells[0] = new StringCell(includedList.get(i));
301       } else {
302         cells[0] = StringCell.TYPE.getMissingCell();
303       }
304 
305       if (i < exludedList.size()) {
306         cells[1] = new StringCell(exludedList.get(i));
307       } else {
308         cells[1] = StringCell.TYPE.getMissingCell();
309       }
310 
311       DataRow newRow = new DefaultRow(key, cells);
312       container.addRowToTable(newRow);
313     }
314 
315     container.close();
316     return container;
317   }
318 
319 
320   /**
321    * {@inheritDoc}
322    */
323   @Override
324   protected void reset() {
325     powerSetGenerator = null;
326       m_lastIteration = false;
327       m_iteration = 0;
328     m_powerSetSize = 0;
329   }
330 
331 
332 
```

```
333   /**
334    * {@inheritDoc}
335    */
336   @Override
337   public boolean terminateLoop() {
338       return m_lastIteration;
339   }
340
341   /**
342      * Creates the specification for the new generated table (
           output port 2)
343      * @return
344      */
345     protected DataTableSpec createOutputTabelSpecTwo() {
346       // the data table spec of the single output table,
347     // the table will have two columns:
348     DataColumnSpec[] allColSpecs = new DataColumnSpec[2];
349     allColSpecs[0] =
350         new DataColumnSpecCreator("Included columns", StringCell
              .TYPE).createSpec();
351     allColSpecs[1] =
352         new DataColumnSpecCreator("Excluded columns", StringCell
              .TYPE).createSpec();
353     return new DataTableSpec(allColSpecs);
354     }
355
356   /**
357    * {@inheritDoc}
358    */
359   @Override
360   protected void validateSettings(final NodeSettingsRO settings)
361       throws InvalidSettingsException {
362     m_exclAsfixed.validateSettings(settings);
363     m_usedColumns.validateSettings(settings);
364     m_minSetSize.validateSettings(settings);
365     m_maxSetSize.validateSettings(settings);
366
367     // workaround for a bug?! (otherwise old data is used!)
368     m_exclAsfixed.loadSettingsFrom(settings);
369     m_minSetSize.loadSettingsFrom(settings);
370     m_maxSetSize.loadSettingsFrom(settings);
371     m_usedColumns.loadSettingsFrom(settings);
372
373     // check if lower bound is smaller than upper bound
374     if (m_maxSetSize.getIntValue() > 0 && (m_minSetSize.
          getIntValue() > m_maxSetSize.getIntValue())) {
375       throw new InvalidSettingsException("Minimal set size can't
              be greater than the maximal set size");
376     }
377
378     // check if ExcludedColumns is not empty when "ExclAsFixed"=
          true
379     if(m_exclAsfixed.getBooleanValue() && m_usedColumns.
          getExcludeList().isEmpty()) {
```

```
380         throw new InvalidSettingsException("Assign columns, which
                have to be 'fixed' to the Excluded List");
381       }
382
383       // check if lower bound is greater than number of columns
384       if(!m_exclAsfixed.getBooleanValue() && m_minSetSize.
              getIntValue() > m_usedColumns.getIncludeList().size()) {
385         throw new InvalidSettingsException("Minimal set size can't
                  be greater than the number of included colums");
386       } else if(m_exclAsfixed.getBooleanValue() && m_minSetSize.
              getIntValue() > (m_usedColumns.getIncludeList().size() +
              m_usedColumns.getExcludeList().size())) {
387         throw new InvalidSettingsException("Minimal set size can't
                  be greater than the number of all colums");
388       }
389
390       // check if maximal set size exceeds the number of columns
391       if(m_exclAsfixed.getBooleanValue() && (m_maxSetSize.
              getIntValue()>(m_usedColumns.getIncludeList().size()+
              m_usedColumns.getExcludeList().size()))) {
392         throw new InvalidSettingsException("Maximal set size can't
                  be greater than the number of all columns");
393       } else if(!m_exclAsfixed.getBooleanValue() && (m_maxSetSize.
              getIntValue()>m_usedColumns.getIncludeList().size())) {
394         throw new InvalidSettingsException("Maximal set size can't
                  be greater than the number of included columns");
395       }
396
397       // check if lower bound is greater than fixed columns
398       if(m_exclAsfixed.getBooleanValue() && m_usedColumns.
              getExcludeList().size()>1 && m_minSetSize.getIntValue()<
              m_usedColumns.getExcludeList().size()) {
399         throw new InvalidSettingsException("Minimal set size can't
                  be less than the number of the fixed columns");
400       }
401       // check if upper bound is greater than fixed columns
402       if(m_exclAsfixed.getBooleanValue() && m_usedColumns.
              getExcludeList().size()>1 && m_maxSetSize.getIntValue()>0
                && m_maxSetSize.getIntValue()<m_usedColumns.
              getExcludeList().size()) {
403         throw new InvalidSettingsException("Maximal set size can't
                  be less than the number of the fixed columns");
404       }
405
406     }
407
408     /**
409      * {@inheritDoc}
410      */
411     @Override
412     protected void loadValidatedSettingsFrom(final NodeSettingsRO
            settings)
413         throws InvalidSettingsException {
414         m_exclAsfixed.loadSettingsFrom(settings);
```

```
415         m_usedColumns.loadSettingsFrom(settings);
416         m_minSetSize.loadSettingsFrom(settings);
417         m_maxSetSize.loadSettingsFrom(settings);
418     }
419
420     /**
421      * {@inheritDoc}
422      */
423     @Override
424     protected void saveSettingsTo(final NodeSettingsWO settings) {
425       m_exclAsfixed.saveSettingsTo(settings);
426       m_usedColumns.saveSettingsTo(settings);
427       m_minSetSize.saveSettingsTo(settings);
428       m_maxSetSize.saveSettingsTo(settings);
429     }
430
431     /**
432      * {@inheritDoc}
433      */
434     @Override
435     protected void loadInternals(final File nodeInternDir,
436         final ExecutionMonitor exec) throws IOException,
437         CanceledExecutionException {
438         // no op
439     }
440
441     /**
442      * {@inheritDoc}
443      */
444     @Override
445     protected void saveInternals(final File nodeInternDir,
446         final ExecutionMonitor exec) throws IOException,
447         CanceledExecutionException {
448         // no op
449     }
450 }
```

Literaturverzeichnis

[1] Advanced modularity-specialzied label propagation algorithm for detecting communities in networks. *Physica A: Statistical Mechanics and its Applications*.

[2] Daniel Aloise, Gilles Caporossi, Pierre Hansen, Leo Liverti, Sylvain Perron, and Manuel Ruiz. Modularity maximization in networks by variable neighborhood search. *Contemporary Mathematics*, 588:113–128, 2012.

[3] Klaus Backhaus, Bernd Erichson, Wulff Plinke, and Rolf Weiber. *Multivariate Analysemethoden*, volume 11. Springer Verlag, 2006.

[4] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. *Graph Partitioning and Graph Clustering*. American Mathematical Society, 2013.

[5] Udo Bankhofer and Jürgen Vogel. *Datenanalyse und Statistik*, volume 1. Gabler Verlag, 2008.

[6] Thomas Böhm. Accuracy improvement of condition diagnosis of railway switches via external data integration. *6th European Workshop on Structural Health Monitoring*, pages 1–9, 2012.

[7] Bundesrepublik Deutschland. *Eisenbahn-Bau- und Betriebsordnung*, May 1967. Bundesrechtsverordnung.

[8] Erhard Cramer and Udo Kamps. *Grundlagen der Wahrscheinlichkeitsrechnung und Statistik*, volume 3. Springer Verlag, 2014.

[9] DB Netz AG. *Infrastrukturzustands- und entwicklungsbericht 2010*, April 2011.

[10] Ágnes Vathy-Fogarassy and János Abonyi. *Graph-Based Clustering and Data Visualization Algorithms*, volume 1. Springer Verlag, 2013.

[11] Pierre Hansen, Nenad Mladenovic, and José A. Moreno Pérez. Variable neighborhood search: methods and applications. 2008.

[12] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2:283–304, 1998.

[13] Achim Klenke. *Wahrscheinlichkeitstheorie*, volume 3. 2008.

[14] Norbert Kusolitsch. *Maß- und Wahrscheinlichkeitstheorie. Eine Einführung*, volume 1. Springer Verlag, 2011.

[15] Ulrich Matuschek. *Sicherheit des Schienenverkehrs*, volume 2. Springer Verlag, 2013.

[16] MIT. Statistics for applications 18.443. 2009. URL: http://ocw.mit.edu/courses/mathematics/18-443-statistics-for-applications-fall-2003/lecture-notes/lec23.pdf.

[17] M. E. J. Newman and M. Girvan. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*.

[18] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev.*

[19] M. E. J. Newman and M. Girvan. Modularity and community structure in networks. *Physical Review E*.

[20] Olufemi A. Omitaomu. Clustering: Categorical attributes. *Lecture Notes in Data Mining*, pages 143–152, 2006.

[21] R. L. Plackett. Karl pearson and the chi-squared test. *International Statistical Review*, 51:59–72, 1983.

[22] Georg Pólya. Über den zentralen grenzwertsatz der wahrscheinlichkeitsrechnung und das momentenproblem. *Mathematische Zeitschrift*, 8:171–181.

[23] Wolfang Rausch. Diagnosesystem für weichen als grundlage für eine optimierte instandhaltungsplanung. *Symposium: Moderne Instandhaltungsverfahren für Weichen - Qualitätsansrprüche - Wirtschaftlichkeit*.

[24] Hana Rezanková. Cluster analysis and categorical data. *Statistika*, 89:216–232, 2009.

[25] Klaus D. Schmidt. *Maß und Wahrscheinlichkeit*, volume 2. Springer Verlag, 2011.

[26] Madeleine Seeland. *Structural Graph Clustering: Scalable Methods and Applications for Graph Classification and Regression*. dissertation, Technische Universität München, 2014. URL: https://mediatum.ub.tum.de/node?id=1212457.

[27] Shokri Z. Selim and M. A. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:81–87, 1984.

[28] Horst Stoll and Bernhard Bollrath. Weichendiagnosesystem sidis w. *Signal + Draht*, 94:26–29, 2002.

[29] Verein Deutscher Ingenieure. *Plant Asset Management (PAM) in der Prozessindustrie*, 1. edition, June 2008. VDI/VDE-Richtlinien.

Erklärung

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

Erlangen, den 16. März 2015