UNIVERSITÄT
BAYREUTH

Datenbasierte Aktivitätserkennung mit Process Mining auf Sensordaten

Dissertation

zur Erlangung des Grades eines Doktors der Wirtschaftswissenschaft

der Rechts- und Wirtschaftswissenschaftlichen Fakultät

der Universität Bayreuth

vorgelegt

von

Dominik Janssen

aus

Schotten

Dekan:	Prof. Dr. Claas Christian Germelmann
Erstberichterstatter:	Prof. Dr. Agnes Koschmider
Zweitberichterstatter:	Prof. Dr. Maximlian Röglinger
Tag der mündlichen Prüfung:	10.07.2025

Abstract

Die zahlreichen Anwendungsmöglichkeiten von Process Mining ermöglichen es, Prozesse aus Rohdaten zu entdecken. In jüngster Zeit hat sich die Anwendung von Process Mining auf unstrukturierte Daten, insbesondere Sensordaten aus IoT-Systemen, als vielversprechendes Forschungsfeld erwiesen (Koschmider et al. 2024). Sensordaten ermöglichen es, Engpässe in IoT-basierten Prozessen (z.B. IoT Fabriken) zu identifizieren und fundierte Entscheidungen zu treffen. Die Anwendung von Process Mining auf Sensordaten ist jedoch ohne mehrere Vorverarbeitungsschritte nicht direkt möglich: Roh-Sensordaten müssen zunächst in diskrete Ereignisse abstrahiert werden. Anschließend werden diese in bedeutungsvolle Ereignisse umgewandelt, um die gewünschten Analyseziele zu erreichen.

Es existiert ein großes Potenzial, aus unstrukturierten Daten Prozessabläufe zu extrahieren. Die große Herausforderung besteht allerdings darin, dass bestehende Techniken im Bereich des Process Minings davon ausgehen, dass Ereignisdaten diskret und auf einer relativ hohen Ebene (d. h. nahe an Geschäftsprozessaktivitäten) vorliegen. In vielen Fällen befinden sich die während der Ausführung eines Prozesses erzeugten Sensordaten jedoch auf einer viel niedrigeren Abstraktionsebene. Die steigende Verfügbarkeit an IoT-Sensordaten in Bereichen wie Smart Cities, Smart Factories oder Smart Homes macht die Verfügbarkeit von effizienten Analysemethoden notwendig.

Die folgende Dissertation beschäftigt sich mit Methoden, die eine effiziente Verarbeitung von unstrukturierten Daten für das Process Mining ermöglichen. Zunächst werden grundlegende Begrifflichkeiten erklärt. Der Hauptteil der Dissertation bildet das sogenannte Framework. Dies ist eine vorgeschlagene und implementierte Methode, wie man aus Sensordaten Aktivitäten erkennen kann. Das vorgeschlagene Framework wird im Evaluationsteil dieser Arbeit an verschiedenen Datensätzen getestet.

Im Anhang dieser Arbeit befinden sich die zu diesem Thema relevanten Publikationen, die die aus dem Hauptteil aufgegriffenen Konzepte vertiefen. Zu diesen relevanten Arbeiten gehört eine umfassende Literaturstudie, welche den aktuellen Stand der Forschung zu Process Mining auf Sensordaten untersucht. Dabei wurden 36 relevante Arbeiten identifiziert, die als Grundlage zur Strukturierung des Forschungsfeldes dienen und Best Practices sowie zukünftige Forschungsperspektiven aufzeigen. Insbesondere die Wahl geeigneter Sensortypen, die Aspekte des Process Mining auf Sensordaten sowie die Verbesserung von Entwurfs- und Evaluationsmethoden werden thematisiert. Es werden in den Publikationen zwei Anwendungsfälle betrachtet. Zum einen Aktivitätserkennung auf getragenen Sensoren und zum anderen Aktivitätserkennung auf stationären Sensoren.

Bei der Analyse von Daten aus Inertialsensoren (IMU) werden Sensoren zur Erkennung menschlicher Aktivitäten und Anomalien genutzt. Die unstrukturierte und kontinuierliche Art dieser Daten erfordert eine effiziente Vorverarbeitung, um sie für Process Mining Algorithmen nutzbar zu machen. Die vorgeschlagene Methode kombiniert Zeitreihen-Segmentierung mit Convolutional Neural Networks (CNNs), um Aktivitäten des täglichen Lebens aus Sensordaten abzuleiten. Die Evaluierung zeigt, dass CNNs besonders gut geeignet sind, wenn Fenstergrößen vorher bekannt sind oder mit einer gleitenden Fensterstrategie angepasst werden. Für die stationären Sensoren wird eine Methode vorgestellt, die Sensordaten in Ereignisdaten für Process Mining Methoden übersetzt. Dabei kommen Vektorisierungs- und Clustering-Methoden zum Einsatz, um Aktivitäten aus Sensordaten abzuleiten. Die Evaluierung anhand eines öffentlich verfügbaren Smart-Home-Datensatzes zeigt, dass sich mit diesem Ansatz auch aus ursprünglich qualitativ niedrigen Sensordaten verwertbare Prozessmodelle ableiten lassen. Der vorgeschlagene Ansatz kombiniert Process Discovery mit unüberwachten Lernmethoden. Ein Feedback-Mechanismus mit Sensitivitätsanalyse verbessert dabei die Qualität der erstellten Prozessmodelle automatisch.

Da nicht alle realen Daten qualitativ hochwertig sind, um für ein effizientes Training der Auswertungsmethoden genutzt werden zu können, wird ein Generator zur Erzeugung synthetischer (Sensor-)Ereignislogs vorgestellt. Dieses Werkzeug ermöglicht eine nicht-deterministische Generierung von Ereignislogs, wobei gezielt Rauschen

hinzugefügt und Prozesse mit IoT-Daten angereichert werden können. Dies erleichtert die Anwendung von Process Mining in experimentellen Umgebungen mit Sensordaten.

Insgesamt trägt diese Arbeit dazu bei, die Integration von Process Mining und Sensordaten methodisch zu verbessern, bestehende Herausforderungen zu adressieren und neue Anwendungsfelder zu erschließen.

Diese Dissertation stellt ein neuartiges Framework vor, das es ermöglicht, Sensordaten in diskrete Ereignisdaten auf höherer Abstraktionsebene zu übersetzen. Dies ermöglicht es, bestehende Process-Mining-Techniken auf die Daten anzuwenden und Prozessmodelle aus Sensordaten zu erstellen. Die vorgestellte Methode erkennt in den beobachteten Sensordaten Aktivitäten, indem unüberwachtes Lernen in Form von Clustering angewendet wird. Darüber hinaus werden die entdeckten Prozessmodelle dahingehend verfeinert, dass auch Aktivitäten gleichzeitig stattfinden können. Der Ansatz wird an verschiedenen öffentlich und nicht-öffentlich zugänglichen Testdaten evaluiert und die Ausdrucksmächtigkeit der Ergebnisse anhand verschiedener Qualitätsmaße evaluiert. Die Ergebnisse zeigen, dass die Anwendung des vorgestellten Frameworks die Verarbeitung von Daten effizient erlaubt, die sich bisher nicht für Process Mining Verfahren eigneten.

Inhaltsverzeichnis

Abkürzungen und Symbole

Abkürzungen

ANN	Artificial Neural Network
BAcc	Balanced Accuracy
BMU	Best Matching Unit
BPM	Business Process Management
BPMN	Business Process Model and Notation
C – net	Causal Net
CASAS	Center of Advanced Studies in Adaptive System
CEP	Complex Event Processing
CNN	Convolutional Neural Network
CPPS	Cyber-Physisches Produktionssystem
CPN	Colored Petri-Net
CSV	Comma-Separated Values
dB	Dezibel
DBScan	Density-Based Spatial Clustering of Applications with Noise
DFG	Directly-Follows Graph
DL	Deep Learning
DT	Decision Tree
EM	Expectation-Maximization
ERP	Enterprise Resource Planning

GUI	Graphical User Interface
HM	Heuristic Miner
IMU	Inertial Measurement Unit
IoT	Internet of Things
K	Kelvin
LSTM	Long Short-Term Memory
M	Motion
MD – RISE	Model-driven Runtime State IdEntification
Mic	Microphone
MJS	Mean Jaccard Similarity
ML	Maschinelles Lernen
MLP	Multilayer Perceptron
N	Newton
OCEL	Object-Centric Event Log
P	Pressure
Pa	Pascal
PELT	Pruned Exact Linear Time
PM	Process Mining
Pm4Py	Process Mining for Python
PNG	Portable Network Graphics
PNML	Petri Net Markup Language
ppm	Parts per Million
RQ	Research Question
SOM	Self-Organising Map
SVM	Support Vector Machine
T	Temperature

TPE	Tree of Parzen Estimators
UML	Unified Modeling Language
WF – Netz	Workflow-Netz
WoS	Web of Science
WMS	Warehouse Management Systems
XES	eXtensible Event Stream
XOR	eXclusive OR

Symbole und Variablen

e	Ereignis
$E_{\mathcal{L}}$	Menge aller Ereignisse im Ereignislog
$\mathcal{E}$	Entität
F	Menge aller Flussrelationen (Petri-Netz)
i	Eingangsstelle (Petri-Netz)
o	Ausgangsstelle (Petri-Netz)
$\mathcal{L}$	Ereignislog
$\mathcal{L}_{\mathcal{S}}$	Sensorereignislog
L	Menge aller Orte
S	Menge aller Sensoren
P	Menge aller Stellen (Petri-Netz)
t^*	Hilfstransition (Petri-Netz)
T	Menge aller Transitionen (Petri-Netz)
$\mathcal{T}$	Menge aller Traces
τ	Trace

1 Einführung

1.1 Einleitung

Durch die immer weiter fortschreitende Verbreitung von Geräten und Sensoren aus dem Internet der Dinge (Internet of Things (IoT)) wird die Auswertung von Sensordaten immer relevanter. Der Arbeitsalltag in Unternehmen ändert sich durch eine stetig wachsende Zahl miteinander verbundener Geräte und Maschinen; so ist im Rahmen von (semi)autonomen Produktionsprozessen die korrekte Auswertung von Sensordaten essenziell. Im Gesundheitsbereich können Sensoren zur Patientenüberwachung genutzt werden, aber auch in Verbindung mit Robotik bei Operationen assistieren. In Privathäusern gibt es vermehrt Sensoren, die das Raumklima überwachen und die Aufenthaltsorte der Einwohner verfolgen können. Städte und Gemeinden werden in Zukunft vermehrt auf Sensordaten zurückgreifen, um eine effiziente Steuerung von Energie und Verkehr zu ermöglichen. All diese Teilbereiche des Internets der Dinge haben gemeinsam, dass viele Daten an unterschiedlichen Stellen anfallen. Um diese Daten weiter zu verarbeiten, bedarf es effizienter Analysetechniken, die es erlauben, wertvolle und neue Erkenntnisse aus den Daten zu liefern.

Mögliche Anwendungsszenarien sind alle Umgebungen, in denen die Bewegung von Objekten oder Personen (Entitäten) durch Bewegungssensoren, Lichtschranken oder ähnliche Arten von Sensoren verfolgt wird, die nur die Abwesenheit oder Anwesenheit einer Person oder eines Objekts erkennen und nicht zwischen verschiedenen beobachteten Entitäten unterscheiden können. Da eine eindeutige Identifizierbarkeit der Entitäten eine Vereinfachung der Problemstellung darstellt, funktioniert das vorgestellte Framework ebenfalls in Umgebungen, in denen eine eindeutige Identifikation der Entitäten gegeben ist.

Techniken aus dem Bereich Process Mining in Verbindung mit effizienter Datenvorverarbeitung bieten genau diese Möglichkeit, und damit wertvolle Einblicke in

das zugrunde liegende Verhalten: Prozesse können aus den Rohdaten identifiziert werden, die von Systemen aufgezeichnet wurden. Eine Identifizierung von Routinen, Engpässen oder Anomalien wird in diesen Prozessen sichtbar. Die Techniken des Process Mining zielen darauf ab, die in Informationssystemen gespeicherten Verhaltensdaten zu nutzen, um die Ausführung von Prozessen zu unterstützen und Prozessmodelle zu erstellen (van der Aalst 2016b). Im Allgemeinen stützt sich das Process Mining auf diskrete Ereignisdaten, von denen in der Regel angenommen wird, dass sie auf der Unternehmensebene erfasst wurden, d. h. die Ereignisdaten stehen in direktem Zusammenhang mit Geschäftsprozessaktivitäten. Die Ebene, auf der die Ereignisdaten innerhalb von Informationssystemen erfasst werden, liegt jedoch häufig auf einer viel niedrigeren Abstraktionsebene, das heißt, dass zwischen einem Sensorereignis alleine keine unmittelbare Beziehung zu einer Aktivität ermittelt werden kann. Sofern Prozesse oft genug wiederholt und in gleicher Weise ablaufen, eignen sich diese gut für die Erstellung von Prozessmodellen mithilfe etablierter Process Mining Methoden. Dies ist jedoch für Prozesse bei realen Szenarien oft nicht der Fall. Sofern es große Variationen in der Reihenfolge der Aktivitäten gibt und Prozesse nur einer ungefähren Prozessdefinition folgen, spricht man von unstrukturierten Prozessen, bei denen die klassischen Process Mining Verfahren nicht immer optimale Lösungen liefern (Diamantini et al. 2016). Durch das Einbeziehen von menschlichem Verhalten und der Natur der Unstetigkeit von menschlichen Abläufen ergeben sich immer wieder kleine und große Abweichungen vom Standard-Prozessablauf. Prozesse, wie sie in der Realität vorkommen, sind häufig unstrukturierter Art (Diamantini et al. 2016, Oberweis 2013). Je größer dieser menschliche Beitrag an den Prozessausführungen, desto komplexer und unübersichtlicher werden die resultierenden Prozessmodelle (Dogan et al. 2019, Dimaggio et al. 2016). Die resultierenden Prozessmodelle sind (ohne effiziente Vorverarbeitung der unstrukturierten Daten) üblicherweise unübersichtliche Modelle, deren Einsatz für weitergehende Analysen nicht immer hilfreich ist (Diamantini et al. 2016).

Um detaillierte Einblicke in die zugrunde liegenden Prozesse mithilfe von Process Mining Verfahren zu gewinnen, müssen demnach eine Reihe von Herausforderungen adressiert werden.

Die Analyse der Daten eines einzelnen Sensors ist in der Regel wenig komplex. Deutlich anspruchsvoller ist die Analyse, wenn der einzelne Sensor Teil eines größeren Netzwerks ist. In solchen Fällen können bereits kleinere Störungen, etwa bei der

Kommunikation zwischen Geräten oder bei der Übergabe von Datenpaketen, die nachgelagerte Prozessanalyse erheblich beeinträchtigen. Aufgrund der Komplexität eines solchen Sensorverbunds ist es daher trotz automatisierter Analyseverfahren unerlässlich, das Fachwissen von Domänenexperten in die Auswertung einzubeziehen. Nur sie können sicherstellen, dass die aus den aufgezeichneten Ereignisdaten abgeleiteten Prozessmodelle korrekt interpretiert und als aussagekräftig validiert werden.

In Umgebungen, in denen es gewünscht ist, häufige Verhaltensmuster oder abnormales Verhalten zu finden, bietet die vorgeschlagene Methode einen neuartigen Ansatz, der Sensordaten zu einem Prozessmodell abstrahiert. Eine der größten Herausforderungen ist, dass in IoT-Umgebungen oft keine streng strukturierten Prozesse existieren. Die Identifizierung unterschiedlicher Entitäten ist ebenso oft unklar wie der Start und Anfang eines Falls (Janiesch et al. 2020). Ohne effiziente Verarbeitung der Rohdaten würde die Entdeckung von Prozessen zu kaum verständlichen Prozessmodellen führen, die für den Benutzer nur von geringem Nutzen sind.

Diese Dissertationsschrift schlägt einen Ansatz zur Entdeckung von Prozessen mithilfe von ungelabelten Sensordaten vor. Der entwickelte Ansatz wird an verschiedenen Szenarien mit verschiedenen Datensätzen evaluiert. Der Hauptfokus liegt auf dem Erkennen von menschlichen Aktivitäten in Smart Home-Umgebungen. Der Ansatz ist jedoch nicht auf eine bestimmte Domäne beschränkt. Stattdessen lässt sich der Ansatz auf jede IoT-Umgebung erweitern, die Daten von Sensoren bereitstellt.

Der Ansatz wird anhand des öffentlich zugänglichen CASAS-Datensatzes (Rashidi et al. 2007) evaluiert und vergleicht mehrere Clustering-Methoden. Die erzielten Ergebnisse sind vielversprechend und deuten darauf hin, dass die Verwendung einer Feedbackschleife und der automatischen Anpassung von Modellparametern brauchbare Ergebnisse liefert.

Das hier vorgestellte Framework ist das erste seiner Art. Bisher gab es keine Technik, die ungelabelte Sensordaten in ihrer Rohform als Input genutzt hat und mithilfe von Process Mining Verfahren brauchbare Prozessmodelle entdeckt hat. Die Dissertation untersucht Ideen, die über die vorhandenen Arbeiten zur direkten Entdeckung von Prozessen in IoT-Daten hinausgehen (Koschmider and Moreira 2018, Seiger et al. 2023, Senderovich et al. 2016).

1.2 Motivation

Sinnvolle Erkenntnisse aus Sensordaten zu gewinnen, ist von enormem Wert. Es gibt sowohl im industriellen Bereich als auch im Gesundheitssektor zahlreiche Szenarien, bei denen eine hohe Menge an Sensordaten generiert wird. In diesen Domänen besteht ein großer Bedarf, die anfallenden Daten effizient zu analysieren. Die Zielsetzung und Motivation für die Auswertung von Sensordaten sind durch das breite Einsatzspektrum von Sensoren quasi unbeschränkt; auszugsweise werden im Folgenden einige Punkte genannt:

Prozessabweichungen: Durch den Einbezug von Sensordaten, können Abweichungen vom Regelprozess in Echtzeit entdeckt werden und frühzeitig Maßnahmen eingeleitet werden.

Predictive Maintenance: Probleme mit Maschinen können mithilfe von Sensordaten erkannt werden, bevor es zu kostspieligen Ausfällen kommt. Durch den Einbezug von Process Mining beschränken sich die Vorhersagemodelle nicht nur auf die jeweilige Maschine, es können auch vor- und nachgelagerte Prozessschritte mit einbezogen werden und verbessert.

Prozessvorhersage: Sind von einem Prozess viele der möglichen Ausführungspfade bekannt, können durch den Einbezug von Sensordaten Vorhersagen getroffen werden, welcher Prozessschritt mit hoher Wahrscheinlichkeit als nächstes eintreten wird.

Assisted Daily Living: Im Gesundheitsbereich spielen Smart Homes im Bezug von *Assisted Daily Living* eine wichtige Rolle. So ist das Ziel in mit Sensoren ausgestatteten Wohnräumen, es älteren Menschen länger als bisher zu ermöglichen eigenständig zu leben (Suryadevara et al. 2013, Ransing and Rajput 2015) und dies möglichst kostengünstig zu tun (Demiris et al. 2008). Je weniger invasiv die Beobachtung der betreffenden Person ist, desto einfacher ist es für die Personen ihren Alltagsgeschäften gewohnt weiter nachzugehen (Jie et al. 2013). Bei älteren Menschen können Abweichungen von einer etablierten Tagesroutine ein Frühindikator auf medizinische Probleme sein (Demiris et al. 2008). Durch das kontinuierliche minimal invasive Beobachten von Personen im Smart Home können reguläre Tagesabläufe gefunden werden

und Abweichungen früh und zuverlässig erkannt werden. Die große Herausforderung besteht darin, Abweichungen korrekt identifizieren zu können und Falsch-Klassifizierungen (*false Positive* und *false Negative*) zu vermeiden (Jie et al. 2013).

Wie in Kapitel 1.1 beschrieben, stellt die Anwendung von Process Mining Verfahren auf Prozesse mit menschlicher Beteiligung bereits eine Herausforderung dar. Sollen reine menschliche Abläufe mit Process Mining beschrieben werden, ergibt sich eine noch größere Herausforderung. Die Ausführungspfade von Prozessen sind durch keine strenge Prozessbeschreibung eingeschränkt. Variationen der Prozessabläufe zwischen einzelnen Tagen können ebenso häufig auftreten wie von einer auf die andere Stunde. Weder die Reihenfolge der Aktivitätsausführungen noch die Menge an möglichen Aktivitäten ist definiert. Vorarbeiten dazu finden sich unter anderem bei (Nazerfard 2018, Lu et al. 2016).

Um etablierte Process Mining Methoden auf unstrukturierte Sensordaten anwenden zu können, bedarf es aufwendiger Vorverarbeitungsschritte, die in dieser Dissertationsschrift aufgezeigt werden.

Sensordaten setzen sich meist aus verschiedenen Quellen zusammen, wie in Abbildung 1.1 für ein Smart Home und in Abbildung 1.3 für eine Smart Factory beispielhaft dargestellt. Die aufgezeichneten Daten liegen häufig, wie in den Tabellen 1.1b und 1.3b dargestellt, ohne jegliche Struktur vor. Diese gilt es, so aufzubereiten, um ein strukturiertes Prozessmodell zu erhalten, wie in Abbildung 1.2 und Abbildung 1.4 gezeigt. Um Prozessdaten auf der Grundlage von Sensor-Ereignisdaten auswerten zu können, sind daher eine effiziente Vorverarbeitung und strukturgebende Datenabstraktions-/Aggregationstechniken erforderlich. Es gibt sechs Ziele, die dabei berücksichtigt werden müssen:

Datenbereinigung: Die Phase der Datenbereinigung ist innerhalb der Prozessanalyse-Pipeline der Schritt, der den größten Einfluss auf das Ergebnis hat. Je gewissenhafter die Daten aufbereitet werden, desto besser und aussagekräftiger werden alle weiteren Auswertungsschritte der Prozessanalyse und damit auch das Endergebnis. Durch das Entfernen von Rauschen, kann sichergestellt werden, dass es keine negativen Auswirkungen auf das Ergebnis der Prozessanalyse gibt. Der Ansatz erkennt, ob parallel ausgelöste Sensoren durch die Bewegung einer einzelnen Person erklärt werden können und somit Sinn ergeben. Wenn dies nicht der Fall ist, werden sie von der weiteren Auswertung ausgeschlossen.

(a) IoT-Sensoren

Zeitstempel	Sensor ID	Wert	Einheit
...	...	...	...
2024-01-01 17:08:53	M_1	1	-
2024-01-01 17:09:23	T_1	293	K
2024-01-01 17:09:25	M_2	1	-
2024-01-01 17:09:28	M_1	0	-
2024-01-01 17:09:56	M_3	1	-
2024-01-01 17:10:42	M_2	0	-
2024-01-01 17:11:56	M_3	0	-
...	...	...	...

(b) Sensorereignislog

Abbildung 1.1: Anwendungsbeispiel Smart Home

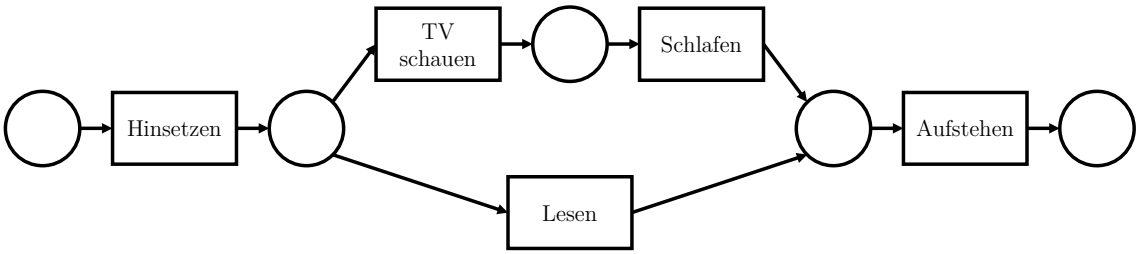


Abbildung 1.2: Beispielprozess in einem Smart Home

(a) IoT-Sensoren

Zeitstempel	Sensor ID	Wert	Einheit
...	...	...	
2024-01-01 11:05:57	$\texttt{Gas}_1$	780	ppm
2024-01-01 11:06:21	$\texttt{P}_1$	1200	N
2024-01-01 11:06:22	$\texttt{T}_1$	465	K
2024-01-01 11:06:24	$\texttt{Gas}_1$	803	ppm
2024-01-01 11:06:36	$\texttt{M}_1$	1	-
2024-01-01 11:07:02	$\texttt{Mic}_1$	104	dB(A)
2024-01-01 11:07:46	$\texttt{M}_1$	0	-
...	...	...	

(b) Sensorereignislog

Abbildung 1.3: Anwendungsbeispiel Smart Factory

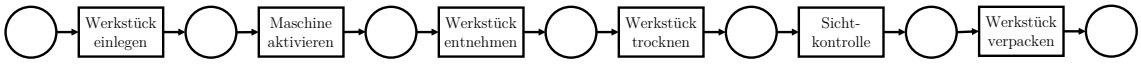


Abbildung 1.4: Einfacher Beispielprozess in einer Smart Factory

Fall-ID: Die Entdeckung von Strukturen oder Routinen in den Ereignisdaten von Sensordaten erfordert die Einbeziehung von Process Mining-Konzepten zu Beginn der Datenanalyse-Pipeline. In dem Ansatz wird der Begriff der Fall-ID und der *Trace* von Anfang an bei der Vorverarbeitung von Sensorereignisdaten berücksichtigt.

Modulare Architektur: Das Framework für das Process Mining auf Sensordaten hat eine modulare Architektur. Jedes Modul der Architektur ist in sich geschlossen und kann mit neuen Techniken (z. B. ML-Techniken zur Datenbereinigung oder Datenaggregation) erweitert werden. Darüber hinaus entlastet ein automatischer Ansatz den Domänenexpterten von der Datenbeschriftung. Der Ansatz reduziert den Aufwand der Datenbeschriftung und trägt somit zu einem (semi)automatischen Ansatz bei.

Unüberwachtes Lernen: Der Aufwand für die Kennzeichnung von Aktivitäten soll minimal sein. Um dies zu erreichen, werden den Aktivitäten zuerst Pseudolabel zugewiesen. Erst nachdem die optimale Kombination aus den Modellparametern gegeben eines Zielfunktionswertes gefunden wurde, müssen die Cluster von einem Domänenexperte gelabelt werden.

Evaluationsfeedback: Die Qualitätsanalyse des Prozessmodells ermöglicht eine Einschätzung zur Effizienz der Rauschfilterung und des Clusterings. Die Bewertung der Qualität von Filterung und Clustering ist herausfordernd, wenn man sie isoliert betrachtet. Dies ist ein wesentlicher Schritt zur Verbesserung der Clustering-Techniken, die in dem Ansatz, der in dieser Dissertation vorgestellt wird, für die Aktivitätserkennung verwendet wird.

Feedback-Schleife: Bei der Verwendung von unüberwachten Clusteringverfahren besteht die Herausforderung darin, dass die Anzahl der Cluster (normalerweise) a priori unbekannt ist. Dies erfordert einen Ansatz, bei dem die Aggregation von Prozessaktivitäten zu Clustern im Nachgang bewertet werden kann, um zu beurteilen, ob angemessen geclustert wurde. Zu diesem Zweck enthält der Ansatz eine Feedback-Schleife: Der Algorithmus bewertet das resultierende Modell und liefert ein Qualitätsmaß, das als Grundlage für die Parameterauswahl und damit der Clusteranzahl der nächsten Iteration dient. Der Ansatz misst, wie empfindlich das entdeckte Prozessmodell gegenüber Veränderungen,

die sich gegenseitig beeinflussen, Parametern für die Abstraktion von Sensorereignisdaten zu Prozessmodellen ist. Dies ermöglicht eine automatische Anpassung an verschiedene Szenarien und die automatische Suche nach einer optimalen Kombination von (Hyper)parametern. Jeder Schritt des Frameworks (z. B. Clustering inklusive Rauschfilterung) kann individuell für den spezifischen Zweck der Anwendungsdomäne konfiguriert werden. Hierdurch kann das vorgestellte Framework auch auf neue Sensordatensätze angewendet werden, ohne größere Anpassungen am Modell vorzunehmen.

Im Vergleich zu eng verwandten Arbeiten auf diesem Gebiet (Cameranesi et al. 2020, Dogan et al. 2019, 2020, Koschmider and Moreira 2018, Seiger et al. 2023, Senderovich et al. 2016, Van Eck et al. 2016) ist der vorgestellte Ansatz der erste, der den gesamten Bereich von den rohen Sensorereignisdaten bis zum fertigen Prozessmodell abdeckt. Teilschritte des Ansatzes wurden bereits in (Koschmider and Moreira 2018, Seiger et al. 2023) erforscht. Allerdings fehlt jegliches qualitativ hochwertiges Feedback in Bezug auf die Qualität des Prozessmodells (z.B. Fitness oder Precision) über die entsprechende Feedbackschleife.

2 Grundlagen

Die für diese Arbeit relevanten Grundlagen und Grundbegriffe werden in den folgenden Unterkapiteln kurz vorgestellt. Hierbei werden an den entsprechenden Stellen die Besonderheiten im Bezug auf den Umgang mit Sensordaten hervorgehoben.

2.1 Process Mining

Process Mining ist eine Methode zur Analyse, Überwachung und Verbesserung von Prozessen auf Basis von Ereignislogs, die bei der Ausführung von Prozessen anfallen. Dabei geht es unter anderem um die Rekonstruktion und Bewertung tatsächlicher Prozessabläufe anhand von erfassten Ereignisdaten. Diese Ereignisdaten sind typischerweise in sogenannten Event Logs gespeichert. Da Process Mining auf wiederkehrenden Abläufen basiert, ist eine ausreichende Anzahl an Prozessausführungen essenziell. Je weniger Prozessausführungen es gibt, desto schwieriger ist es, daraus ein aussagekräftiges Prozessmodell zu erstellen (van der Aalst 2010). Process Mining lässt sich in die folgenden drei Gebiete aufteilen (van der Aalst 2016a):

- Process Discovery
- Conformance Checking
- Enhancement

Für diese Arbeit relevant sind die Gebiete des *Proess Mining* und das *Conformance Checking*. Das *Process Discovery* wird für die Prozessmodellerstellung aus den Sensordaten verwendet und das *Conformance Checking* für die Bewertung der Prozessmodellqualität. Diese werden in den Kapiteln Process Discovery und Conformance Checking genauer beschrieben.

Ereignis

Das Ereignis $e \in E_{\mathcal{L}}$ muss einige Mindestanforderungen erfüllen, damit diese für die Auswertung des in dieser Arbeit vorgestellten Frameworks verwendet werden können: Für jedes Ereignis muss ein Zeitstempel $time(e)$ existieren, welcher die Ordnung der Ereignisse erlaubt. Des Weiteren muss es eine Sensorkennzeichnung $sensor(e) \in S$ geben, die angibt, welcher Sensor aktiviert wurde. Im Fall von ortsbezogener Auswertung müssen Informationen zu dem Sensor vorhanden sein, die sich entweder implizit oder explizit auf einen Ort beziehen (d.h. $location(e) \in L$). Die Ortsangabe kann explizit in Form von Koordinaten (z.B. Breitengrad und Längengrad) oder implizit durch die Angabe von gekennzeichneten Orten zusammen mit einer Abstandsfunktion, die paarweise Abstände zwischen ihnen liefert, erfolgen. Zusätzlich muss das Ereignis, welches im Sensorereignislog (siehe Sensorereignislog) gespeichert wird, Rückschlüsse über den Sensorstatus (z.B. aktiviert, deaktiviert, Temperaturwert usw.) erlauben.

Aktivität

Im klassischen Process Mining besteht eine Aktivität aus einer Menge von Aufgaben, die von einer bestimmten Ressource für einen Fall ausgeführt wird (Laue et al. 2020). Im Kontext dieser Arbeit besteht eine Aktivität aus einer Menge an Sensoraktivierungen. Die Sensoraktivierungen entsprechen im klassischen Process Mining Aufgaben bzw. atomaren Arbeitsschritten.

Abbildung 2.1 zeigt exemplarisch, dass eine Aktivität aus mehreren Sensoraktivierungen in einer bestimmten Reihenfolge besteht. So hat jede Aktivität, die beobachtet werden kann, eine bestimmte Abfolge an Sensoraktivierungen. Hierbei ist es möglich, dass für eine Aktivität mehrere solcher Sensoraktivierungsmuster existieren.

Eine Sensoraktivierung isoliert zu betrachten, lässt nur bedingt Schlüsse auf die zugrunde liegende Aktivität zu. Erst durch das Einbeziehen mehrerer aufeinanderfolgender Sensoraktivierungen können Aktivitäten zuverlässig erkannt werden.

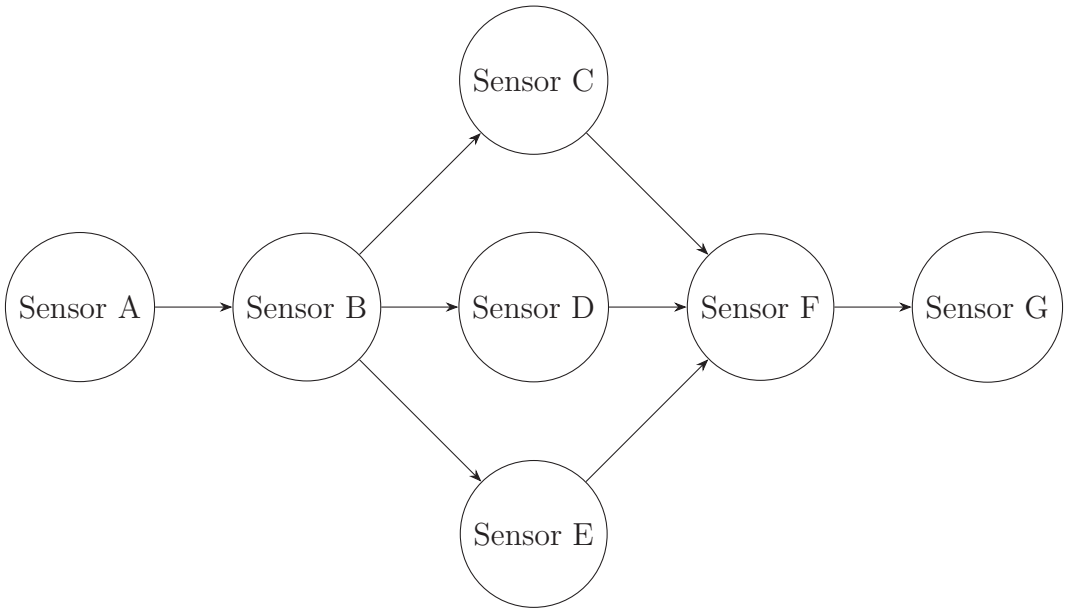


Abbildung 2.1: Beispiel einer Aktivität mit Sensordaten

Trace

Eine Trace τ, auch als *Ablauf* bezeichnet, sind zeitlich geordnete Ereignisse eines Falls (Laue et al. 2020). Im Kontext von Sensordaten ist eine Trace eine Menge an Sensoraktivierungen, die in Form eines Vektors dargestellt sind. Die Beispielaktivität, wie sie in Abbildung 2.1 dargestellt ist, kann durch drei mögliche Traces dargestellt werden:

$$(A, B, C, F, G)$$
$$(A, B, D, F, G)$$
$$(A, B, E, F, G)$$

Diese Darstellung enthält jedoch keine Informationen über die Aktivierungsdauer der einzelnen Sensoren. Um die Zeitkomponente in die Tracevektoren einzubeziehen, ist es möglich, die Tracevektoren zu transformieren. Die detaillierte Vorgehensweise dazu wird in Kapitel 3.3 gezeigt.

Fall

Ein Fall ist eine Instanz einer Trace. Es können mehrere Fälle existieren, die das Muster einer Trace abbilden (van der Aalst 2010).

Tabelle 2.1: Beispiel für einen Ereignislog zu Beispielprozess aus Abbildung 1.4

Ereignis-ID	Fall-ID	Zeitstempel	Ereignis	Status
...	...	...	...	...
1032	242	2025-01-05 08:40:00	`Werkstück einlegen`	Start
1033	242	2025-01-05 08:41:20	`Werkstück einlegen`	Ende
1034	242	2025-01-05 08:41:29	`Maschine aktivieren`	Start
1035	242	2025-01-05 08:42:45	`Maschine aktivieren`	Ende
1036	242	2025-01-05 08:42:47	`Werkstück entnehmen`	Start
1037	242	2025-01-05 08:43:05	`Werkstück entnehmen`	Ende
1038	242	2025-01-05 08:46:03	`Werkstück trocknen`	Start
1039	242	2025-01-05 09:38:00	`Werkstück trocknen`	Ende
1040	242	2025-01-05 09:50:02	`Sichtkontrolle`	Start
1041	242	2025-01-05 09:52:15	`Sichtkontrolle`	Ende
1042	242	2025-01-05 09:52:23	`Werkstück verpacken`	Start
1043	242	2025-01-05 09:53:00	`Werkstück verpacken`	Ende
...	...	...	...	...

Ereignislog

Die Basis für die Analyse von Prozessen im Process Mining bildet das Ereignislog (Van Der Aalst 2012a) auch Ereignisprotokoll genannt (Laue et al. 2020). Die Mindestanforderung an das Ereignislog $\mathcal{L}$ ist das Enthalten eines Zeitstempels, einer Fall-ID und einer Ereignisbezeichnung. Der Zeitstempel kann als absolute Zeitangabe angegeben sein oder durch eine relative Zeitangabe zu anderen Ereignissen dargestellt sein. Ein Ereignislog kann noch weitere Informationen beinhalten, wie zum Beispiel die Ressource, die für die Ausführung der Aktivität des entsprechenden Eintrags verantwortlich ist. Beispielhaft ist in Tabelle 2.1 ein Ereignislog eines fiktiven Produktionsprozesses (aus Abbildung 1.4) dargestellt.

In klassischen Process Mining Anwendungen, meist im geschäftlichen Umfeld, sind Ereignislogs oftmals Auszüge aus ERP- oder WMS-Systemen (Carmona et al. 2018). Mithilfe dieser Systeme ist eine Aufzeichnung von auftretenden Aktivitäten in hoher

Qualität möglich. Die genaue Qualität der Ereignislogs kann jedoch stark variieren und ist oft von vielen Faktoren abhängig (Van Der Aalst et al. 2012).

Im Idealfall sind Ereignisse im Ereignislog sequentiell und vollständig aufgezeichnet. Jeder Eintrag, also jedes Ereignis, im Ereignislog bezieht sich auf einen genauen Prozessschritt und bezieht sich auf einen bestimmten Fall (Van Der Aalst 2012b, Accorsi et al. 2012). In der Realität liegen die Daten jedoch oft in unstrukturierter Form und dezentral vor. Die vorhandenen Daten in ein, für die Process Mining Verfahren geeignetes Ereignislog zu überführen, bedarf weiterer Verarbeitungsschritte (van der Aalst 2016b).

Eine Sonderform des Ereignislogs ist der Sensorereignislog; in diesem werden Ereignisse nicht explizit angegeben, sondern Sensorwerte, die während einer Aktivitätsausführung aufgezeichnet werden, siehe dazu Unterkapitel Sensorereignislog.

Sensorereignislog

Sind nun in einer beobachteten Umgebung Sensoren angebracht, so dass einzelne Prozessschritte mit Sensoren überwacht werden können, ist es möglich, einen Sensorereignislog ($\mathcal{L}_S$) zu erstellen. Der Beispielprozess aus Abbildung 1.4 ist in der Abbildung 2.2 mit zusätzlichen Sensoren an den jeweiligen Prozessschritten dargestellt.

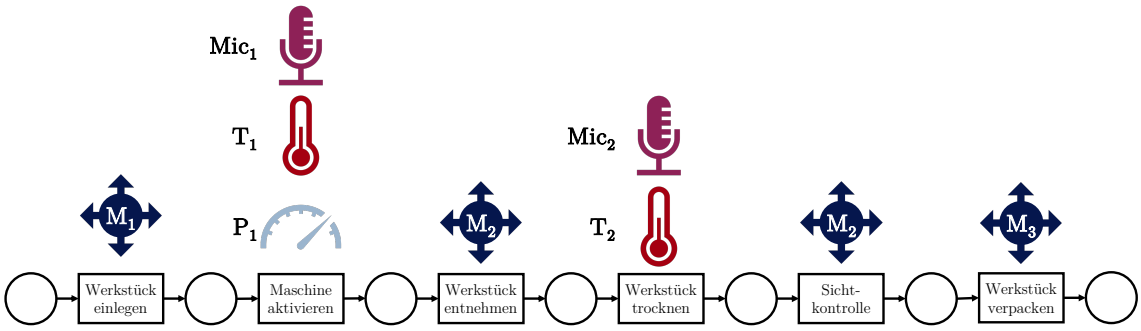


Abbildung 2.2: Erweiterung des Beispielprozesses aus Abbildung 1.4 um Sensoren

In einem Sensorereignislog, wie er beispielhaft in Tabelle 2.2 dargestellt ist, sind üblicherweise keine konkreten Ereignisbezeichnungen zu finden, wie sie normalerweise in gewöhnlichen Ereignislogs vorhanden sind (siehe Tabelle 2.1).

Tabelle 2.2: Beispiel für einen Sensorereignislog zu Beispielprozess aus Abbildung 1.4

E-ID aus Tbl. 2.1	**Ereignis-ID**	**Zeitstempel**	**Sensor-ID**	**Wert**	**Einheit**
...	...	...	...	...	...
–	90431	25-01-05 08:39:55	$\texttt{M}_1$	Inaktiv	
1032	90432	25-01-05 08:40:00	$\texttt{M}_1$	Aktiv	
1033	90433	25-01-05 08:41:20	$\texttt{M}_1$	Inaktiv	
–	90434	25-01-05 08:41:28	$\texttt{P}_1$	0	MPa
–	90435	25-01-05 08:41:28	$\texttt{T}_1$	293	K
–	90436	25-01-05 08:41:28	$\texttt{Mic}_1$	70	dB
1034	90437	25-01-05 08:41:29	$\texttt{P}_1$	5	MPa
1034	90438	25-01-05 08:41:31	$\texttt{T}_1$	330	K
1034	90439	25-01-05 08:41:59	$\texttt{P}_1$	10	MPa
1034	90440	25-01-05 08:42:29	$\texttt{P}_1$	20	MPa
1034	90441	25-01-05 08:42:30	$\texttt{Mic}_1$	90	dB
1034	90442	25-01-05 08:42:40	$\texttt{P}_1$	5	MPa
1034	90443	25-01-05 08:41:31	$\texttt{T}_1$	430	K
1035	90444	25-01-05 08:42:45	$\texttt{P}_1$	0	MPa
1035	90445	25-01-05 08:42:45	$\texttt{Mic}_1$	70	dB
–	90446	25-01-05 08:42:46	$\texttt{M}_2$	Inaktiv	
1036	90447	25-01-05 08:42:47	$\texttt{M}_2$	Aktiv	
1037	90448	25-01-05 08:43:05	$\texttt{M}_2$	Inaktiv	
...	...	...	...	...	...

Die Herausforderung bei Sensorereignislogs ist in Tabelle 2.2 zu sehen:

Uneindeutige Ereignisbeziehung. Wenn man die Einträge des Sensorereignislogs in Tabelle 2.2 mit dem klassischen Ereignislog in Tabelle 2.1 vergleicht, erkennt man in der Spalte „**E-ID aus Tbl. 2.1**“, dass zu vielen Einträgen des Sensorereignislogs sich nur eine Ereignis-ID des ursprünglichen Logs beziehen. Dies verdeutlicht die Herausforderung, dass Ereignisse des Sensorereignislogs für eine aussagekräftige Aktivitätserkennung sinnvoll aggregiert werden müssen.

Unterschiedliche Sensortypen. Es müssen unter Umständen unterschiedliche Sensortypen verarbeitet werden.

Fall ID. Im Beispiel aus Tabelle 2.2 fehlt die Spalte Fall-ID. Bei Aktivitäten die von Sensoren aufgezeichnet werden, ist es möglich, dass keine direkte Zuordnung zu einem Fall zur Verfügung gestellt wird. Eine Fallzuordnung muss dann über heuristische Verfahren ermittelt werden.

Entität

Eine Entität $\mathcal{E}$ ist im betrachteten Kontext eine Person oder ein Objekt, welches sich durch den überwachten Raum bewegt. Die Präsenz, das Verhalten und die Bewegung der Entität beeinflussen die Werte der Sensoren. Diese Ereignisse bzw. Sensorereignisse (siehe Kapitel 2.1) werden in einem Ereignislog (siehe Kapitel 2.1) aufgezeichnet.

Process Discovery

Process Discovery Verfahren ermöglichen es, aus aufgezeichneten Daten Prozessmodelle zu erstellen. Die Datengrundlage bilden hier sogenannte Ereignislogs (siehe Kapitel Ereignislog). Anhand dieser Prozessmodelle können die zugrunde liegenden Prozesse objektiv analysiert werden und ermöglichen einen transparenten Blick auf die tatsächlich ablaufenden Prozesse (Accorsi et al. 2012, van der Aalst 2016b). Mögliche Anwendungsszenarien für *Process Discovery* sind beispielsweise, Engpässe in Prozessen zu finden, Probleme bei der Prozessausführung frühzeitig zu erkennen oder Regelverstöße gegen vorgeschriebene Prozessrichtlinien aufzudecken. Aus den

gewonnenen Erkenntnissen ist es möglich, fundierte Handlungsempfehlungen abzuleiten. Dies ermöglicht es, die erkannten Probleme zu beheben und somit Prozesse zu optimieren (Oberweis 2013, Van Der Aalst et al. 2012, van der Aalst 2016b). In Kapitel 2.4 findet sich eine Auflistung von *Process Discovery* Algorithmen, die für diese Arbeit relevant sind.

Conformance Checking

Die Bewertung von Modellen in Beziehung zu den Ereignislogs fällt in den Bereich des *Conformance Checkings*. Es wird hierbei überprüft, inwiefern das entdeckte Prozessmodell zu den aufgezeichneten Daten passt. Für die Bewertung existieren verschiedene Qualitätsmaße.

Fitness

Die Kennzahl *Fitness* (auch *Recall* genannt) gibt an, wie viele der Traces aus dem Ereignislog vom Prozessmodell dargestellt werden können (Buijs et al. 2012). Der Fitnesswert berechnet sich durch den Quotienten zwischen den Traces, die vom Prozessmodell abgebildet werden, und der Gesamtzahl der im Ereignislog enthaltenen Traces.

$$Fitness = \frac{|Log \cap Modell|}{|Log|} \tag{2.1}$$

Die Fitness eines Modells kann Werte zwischen 0 und 1 annehmen. Ist die Fitness 1, können alle im Ereignislog enthaltenen Traces vom Modell abgeblidet werden. Die Fitness ist 0, wenn keine Trace des Ereignislogs vom Modell abgespielt werden kann.

Eine perfekte Fitness ist jedoch nicht unbedingt erstrebenswert. Selten auftretende Prozessvariationen können das Prozessmodell unnötig verkomplizieren und ein unpräzises Modell wie das *Flower-Modell* besitzt eine perfekte Fitness für gegebene Ereignislogs, ist jedoch meist kein wünschenswertes Prozessmodell. Es hilft, ein Trade-Off mit anderen Kennzahlen zu finden.

Token-Based Replay Eine Vorgehensweise, die Fitness zu berechnen, ist die Token-Based Replay Methode. Hierbei wird versucht, jeden im Ereignislog aufgezeichneten Fall auf dem erstellten Petri-Netz Modell abzuspielen. Für Transitionen,

die erfolgreich aktiviert werden können, wird je eingehender Kante ein Token konsumiert (c) und für jede ausgehende Kante ein Token produziert (p). Fehlende Token in Vorbereichen von Transitionen werden als fehlend (m) gezählt und im Netz verbleibende Token nach Beendigung als verbleibend (r) gezählt. Die Berechnung der Token-Based Fitness für ein gegebenes Modell N und das zugrunde liegende Ereignislog ($\mathcal{L}$) ist in 2.2 dargestellt (Van Der Aalst 2012b).

$$\text{fitness}(\mathcal{L}, N) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma\in\mathcal{L}} \mathcal{L}(\sigma) \times m_{N,\sigma}}{\sum_{\sigma\in\mathcal{L}} \mathcal{L}(\sigma) \times c_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{\sum_{\sigma\in\mathcal{L}} \mathcal{L}(\sigma) \times r_{N,\sigma}}{\sum_{\sigma\in\mathcal{L}} \mathcal{L}(\sigma) \times p_{N,\sigma}}\right) \quad (2.2)$$

Durch die geringe Komplexität bei der Berechnung der Token-Based Fitness kann auch für umfangreiche Ereignislogs und große Prozessmodelle in kurzer Zeit ein Fitnesswert berechnet werden (van der Aalst 2016a). Aufgrund der Einfachheit des Berechnungsansatzes ergeben sich jedoch einige Schwächen. So kann es vorkommen, dass bei großen Abweichungen zwischen Ereignislog und Modell das Petri-Netz mit Tokens übersättigt wird. Diese unpassende Modell/Ereignislog-Kombination wird dann in der Regel mit zu guten Fitnesswerten bewertet (Berti and van der Aalst 2019, Carmona et al. 2018). Modelle mit mehrfach vorkommenden Aktivitätsnamen können bei diesem Verfahren ebenfalls nicht immer korrekt bewertet werden (Van Der Aalst 2012b, van der Aalst 2016a).

Alignment-Based Replay Die zuvor erwähnten Nachteile des Token-Based Replays werden versucht, mit der Alignment-Based Fitnessberechnung auszubessern. Statt mithilfe von Tokens die Fälle im erstellten Modell nachzuspielen, werden beim Alignment-Based Replay sogenannte Alignments gesucht. Ein Alignment ist eine bestmögliche Übereinstimmung zwischen einem zulässigen Pfad durch das erstellte Prozessmodell und einem im Ereignislog aufgezeichneten Fall (van der Aalst 2016a).

$$\text{fitness}(\mathcal{L}, N) = 1 - \frac{\sum_{\sigma\in\mathcal{L}} \delta\left(\lambda^N_{\text{opt}}(\sigma)\right)}{\sum_{\sigma\in\mathcal{L}} \delta(\lambda^N_{\text{worst}}(\sigma))} \quad (2.3)$$

Die Berechnung der Fitness mit dieser Methode ist oftmals genauer. Im Gegensatz zur Token-Based Methode ist das Alignment-Based Verfahren nicht abhängig von Petri-Netzen, sondern kann auf Prozessmodelle in beliebigen Modellierungssprachen verwendet werden. Aufgrund des großen Suchraums ist die Berechnung

des Alignment-Based Replays bei größeren Modellen allerdings sehr zeitaufwendig oder nicht deterministisch.

Precision

Die Precision bewertet, wie viele Traces sich vom Modell abspielen lassen, im Verhältnis zu den tatsächlich im Ereignislog beobachteten Traces (Tax et al. 2018a).

$$Precision = \frac{|Log \cap Modell|}{|Modell|} \tag{2.4}$$

Die Precision eines Modells liegt zwischen 0 und 1. Bei einer Precision von 1 kann das Modell ausschließlich Traces darstellen, die auch im Ereignislog beobachtet wurden. Liegt die Precision bei 0, handelt es sich um ein unpräzises Modell; keine der vom Modell möglichen Traces kommt im Ereignislog vor.

Das in Kapitel 2.1 angesprochene *Flower-Modell* hat generell eine sehr schlechte Precision, da dieses Modell alle möglichen Trace-Abfolgen zulässt, auch wenn diese nicht im Prozessmodell vorkommen (Tax et al. 2018a). Eine perfekte Precision von 1 zu erreichen, ist oft schwierig, jedoch auch nicht unbedingt erstrebenswert (Laue et al. 2020).

Simplicity

Die Simplicity-Kennzahl bewertet, wie komplex, bzw. simpel ein Modell ist. Eine Optimierung dieser Kennzahl erlaubt es, Prozessmodelle zu erstellen, die einfach verständlich und nachvollziehbar sind (Buijs et al. 2012).

Generalisation

Die Maximierung der *Generalisation* eines Prozessmodells steht im diametralen Gegensatz zur Precision. Bei einer hohen *Generalisation* ist auch Verhalten im Prozessmodell erlaubt, welches bisher (noch) nicht aufgetreten ist (Buijs et al. 2012, Laue et al. 2020).

F1-Score

Um zu vermeiden, dass Modelle ausschließlich nach einem Kriterium optimiert werden (sofern man die vorigen genannten Bewertungskriterien als Zielfunktionswert einer Optimierung verwendet), gibt es den F_1-Wert, der Fitness und Precision kombiniert. Der F_1-Score kann zwischen $[0, 1]$ liegen und berechnet sich aus dem harmonischen Mittel zwischen Precision und Fitness (Umer et al. 2017).

$$F_1 = \frac{2}{Fitness^{-1} + Precision^{-1}} \tag{2.5}$$

2.2 Sensoren

Die folgende Beschreibung von Sensoren und deren Klassifizierung ist an Brzychczy et al. (2025) (Anhang B) angelehnt:

Mithilfe von Sensoren können Ereignisse der physischen Welt aufgezeichnet werden. Darunter fallen beispielsweise Bewegung, Temperatur, Druck oder Drehzahlen. Sensoren lassen sich in zwei Gruppen einteilen:

Tragbare Sensoren: Tragbare Sensoren werden oft dazu genutzt, Aktivitäten von Personen zu überwachen. Die Arbeit von Polle et al. (2024) untersucht den Einsatz von tragbaren Sensoren im Gesundheitskontext. Tragbare Sensoren müssen jedoch nicht gezwungenermaßen von Menschen getragen werden. Tiere, Packstücke, Fahrzeuge oder andere Anwendungsszenarien sind denkbar.

Stationäre Sensoren: Stationäre Sensoren werden oft dazu genutzt, Umweltbedingungen zu sammeln. Die Arbeiten Janssen et al. (2020) und Janssen et al. (2026) befassen sich mit stationären Sensoren.

Beide Sensortypen können sowohl diskrete als auch kontinuierliche Zeitreihen aufzeichnen. Für die es jeweils unterschiedliche Herangehensweisen mit ihren eigenen Herausforderungen gibt.

In der Literatur gibt es bereits folgende Arbeiten, die sich mit Process Mining und Sensoren beschäftigen. Dort kommen unterschiedliche Techniken zum Einsatz: Senderovich et al. (2016) untersuchen Sensordaten im Gesundheitswesen. In dem untersuchten Krankenhaus werden die Standorte von Geräten, Krankenhausmitarbeitern

und Patienten aufgezeichnet. Die Verarbeitung der Sensordaten erfolgt durch Aggregation, Erkennung von Interaktionen und anschließende Zuordnung dieser Interaktionen zu Prozessaktivitäten mithilfe von Prozesswissen und Optimierung. In Dogan et al. (2019) werden mithilfe von Bewegungsmeldern die Standorte und Bewegungen von Personen innerhalb von Wohnungen aufgezeichnet, um so Benutzerpfade zu ermitteln. Aus den Aktivierungsdaten der Sensoren werden Ereignislogs erstellt und aus diesen mit Process Mining Verfahren Prozessmodelle erstellt. Ähnliche Verhaltensmuster werden mithilfe von Clusteringverfahren zusammengefasst. Gleichzeitig ausgeführte Prozessaktivitäten können jedoch nicht erkannt werden. Die Publikation von Dogan et al. (2020) erfasst Standortdaten von Kunden in einem Einkaufszentrum mithilfe von Bluetooth-Sensoren. Diese Bewegungspfade der Kunden innerhalb der Verkaufsräume werden mithilfe eines fuzzy-basierten Ansatzes zu ähnlichen Bewegungsabläufen zusammengefasst. Es wird jedoch kein Gesamtprozessmodell aus den Sensorereignisdaten ermittelt.

2.3 Prozessdarstellung

Um Prozesse zu modellieren und grafisch darzustellen, gibt es unterschiedliche Möglichkeiten. Die für diese Arbeit relevanten Darstellungsweisen werden im Folgenden erläutert.

Directly-Follows Graph

Der Directly-Follows Graph (DFG) ist eine beliebte Darstellungsweise von Prozessen in kommerziellen Process Mining Tools (van der Aalst 2019) In einem *DFG* wird ausschließlich betrachtet, welche Aktivitäten aufeinander folgen. In Abbildung 2.3 ist ein Directly-Follows Graph beispielhaft dargestellt. Bei der Betrachtung der Aktivitäten *Werkstück entnehmen* und *Maschine ausschalten* in Abbildung 2.3 wird eine Schwäche des Directly-Follows Graphen offensichtlich: Laut des Directly-Follows Graphen ist eine wiederholte Ausführung (Schleife) der beiden Aktivitäten *Werkstück entnehmen* und *Maschine ausschalten* möglich. Des Weiteren ist es laut Directly-Follows Graphen ein zulässiger Prozessablauf, dass nach *Maschine aktivieren* ausschließlich die Aktivität *Maschine ausschalten* ausgeführt wird und das Werkstück in der Maschine verbleibt. Bei Directly-Follows Graphen muss also immer

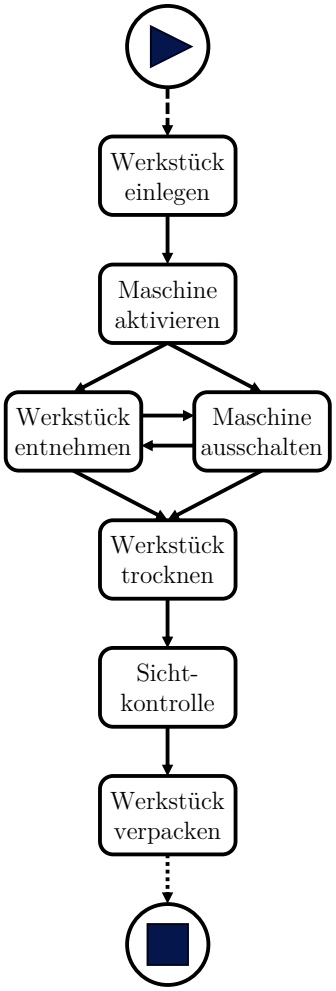


Abbildung 2.3: Beispiel eines Directly-Follows Graphen.

bedacht werden, dass diese Prozessverhalten darstellen, welches nie im Ereignislog auftritt.

Bei der Erstellung eines Directly-Follows Graphen können drei Schwellwerte bestimmt werden

Fallhäufigkeit: Wie oft tritt eine Ausprägung einer *Trace* auf.

Ereignishäufigkeit: Wie oft tritt ein Ereignis auf.

Abfolgehäufigkeit: Wie oft kommt die Abfolge `Ereignis A` > `Ereignis B` im Datensatz vor.

Die simple Erstellung und Darstellungsweise des Directly-Follows Graphen birgt jedoch die Gefahr falscher Schlussfolgerungen (van der Aalst 2019). Es ist in einem

Directly-Follows Graphen nicht möglich, Parallelität zwischen Aktivitäten explizit darzustellen: Folgen auf eine Aktivität mehrere Aktivitäten, ist es nicht möglich zu unterscheiden, ob es sich bei den Folgeaktivitäten um eine exklusive Auswahl handelt oder die Folgeaktivitäten parallel auftreten (Laue et al. 2020). Ein *DFG* ist eine nicht sehr präzise Darstellung eines Prozesses: Der erstellte Directly-Follows Graph erlaubt meist eine größere Variation an Prozessabläufen, die eigentlich in dem Ereignislog vorkommen (Laue et al. 2020). Directly-Follows Graphen sind gut geeignet, einen ersten Überblick über einen Prozess zu erhalten; definitive Schlussfolgerungen aus Directly-Follows Graphen sind jedoch nur bedingt ratsam.

Petri-Netz

Petri-Netze können als eine Darstellungsweise für Prozesse verwendet werden und gehen auf die Arbeit von Carl Adam Petri zurück (Petri 1962). Ursprünglich im Kontext der theoretischen Informatik entwickelt, finden Petri-Netze auch Anwendung bei der Modellierung von Geschäftsprozessen (van Hee et al. 2013). Petri-Netze sind bipartite gerichtete Graphen, bei denen zwischen zwei Arten von Knoten unterschieden wird: Stellen und Transitionen. Die Stellen und Transitionen sind mit gerichteten Kanten verbunden. In der Geschäftsprozessmodellierung stehen Stellen meist für Zustände und Transitionen für Aktivitäten (Laue et al. 2020). Sie sind folgendermaßen definiert (Van der Aalst 1997):

Definition 1 *Petri-Netz.*
Ein Petri-Netz ist ein Tripel (P, T, F) *mit:*

- P*, der endlichen Menge der Stellen*
- T*, der endlichen Mengen der Transitionen*
- $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$*, die Menge von Kanten (Flussrelationen)*

Das Petri-Netz in Abbildung 2.4 zeigt den gleichen Prozess, wie er in Abbildung 2.3 als Directly-Follows Graph dargestellt ist. Man kann in dem Petri-Netz ablesen, dass die Aktivitäten *Werkstück entnehmen* und *Maschine ausschalten* parallel zueinander stattfinden. Dieser Zusammenhang zwischen diesen beiden Aktivitäten ist im Directly-Follows Graph aus Abbildung 2.3 nicht ersichtlich. Gemäß des DFGs

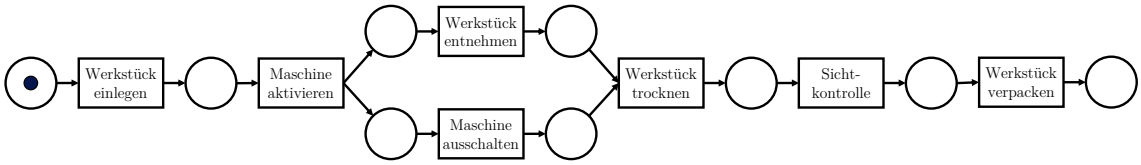


Abbildung 2.4: Beispiel eines Petri-Netzes.

können die Aktivitäten sowohl parallel als auch exklusiv zueinander stattfinden. Das Prozessmodell, modelliert als Petri-Netz, ist in diesem Fall eine präzisere Darstellung des Prozesses.

Workflow-Netze

Workflow-Netze (WF-Netze) sind eine spezielle Art von Petri-Netzen; sie sind wie folgt definiert (Van der Aalst 1997):

Definition 2 *Workflow-Netz.*
Ein Petri-Netz ist ein Workflow-Netz genau dann, wenn:

- *Es eine Stelle i gibt, die keine eingehenden Kanten besitzt.*
- *Es eine Stelle o gibt, die keine ausgehende Kanten besitzt.*
- *Das um die Hilfstransition t^* zwischen o und i erweiterte Netz stark zusammenhängend ist.*

Workflow-Netze sind insbesondere für Business Process Management Anwendungen bzw. der Modellierung von Geschäftsprozessen relevant. Durch die oben genannten Eigenschaften eines Workflow-Netzes gibt es immer genau einen Anfang, zu dem ein Fall initiiert wird. Durch die Bedingung, dass es genau eine Stelle geben muss, die keine ausgehenden Kanten besitzt, hat auch ein einmal initiierter Fall einen fest definierten Endzustand (Van Der Aalst et al. 2011).

Soundness

Nicht jedes Workflow-Netz stellt einen korrekt ausführbaren Prozess dar. In einem Prozessmodell können folgende Probleme auftreten:

- Deadlocks
- Aktivitäten, die niemals ausgeführt werden können
- Endlosschleifen
- Übriggebliebene Tokens nach Prozessende

Um sicherzustellen, dass ein Workflow-Netz einen sinnvollen und fehlerfreien Prozess beschreibt, existiert das Konzept der Soundness. Die Soundness Eigenschaft garantiert, dass die oben aufgelisteten Probleme nicht auftreten können.

Im Folgenden wird die formale Definition von Soundness gegeben (van der Aalst 2016a).

Definition 3 *Soundness.*
Ein Workflow-Netz $N = (P, T, F)$ mit der Eingangsstelle i und der Ausgangsstelle o wird als sound bezeichnet, wenn gilt:

Sicher: *Das Workflow-Netz ist sicher. Es gibt in keinem Schaltzustand eine Stelle, die mehr als ein Token markiert.*

Ordnungsgemäße Beendigung: *Für jede erreichbare Markierung M muss gelten: Wenn die Ausgangsstelle o mit einem Token markiert ist, ist dieser Token der einzigen Token im gesamten Netz N.*

Möglichkeit der Beendigung: *Aus jeder erreichbaren Markierung M muss es möglich sein die Ausgangsstelle o zu erreichen.*

Keine toten Transitionen: *Keine Transition im Workflow-Netz N ist tot.*

Causal Net

Causal Nets (C-nets) sind eine Modellierungssprache, die speziell für Process Mining, insbesondere für Process Discovery, entwickelt wurde.

Ein Causal Net ist ein gerichteter Graph, Knoten repräsentieren die Aktivitäten und die Kanten stehen für kausale Abhängigkeiten zwischen diesen Aktivitäten. Ein zentraler Bestandteil von C-nets ist die Verwendung von Input- und Output-Bindungen. Diese spezifizieren für jede Aktivität, welche Kombinationen von Vorgänger- und Nachfolgeraktivitäten zulässig sind. Hiermit lassen sich Kontrollflusskonstrukte wie

AND-, OR- oder XOR-Verzweigungen direkt über die Bindungsstruktur abbilden. Ein Verwenden zusätzlicher Modellierungselemente wie Stellen oder Gateways ist damit nicht mehr notwendig (Van Der Aalst et al. 2011).

Abhängigkeitsgraph

Ein Abhängigkeitsgraph (Dependency Graph) zeigt Abhängigkeiten zwischen Aktivitäten an. Die Aktivitäten werden als Knoten dargestellt, und gerichtete Kanten zeigen, welche Aktivitäten voneinander abhängen. Die Kantengewichte sind die sogenannten Abhängigkeitsmaße (Definition5.

Definition 4 *Abhängigkeitsgraph.*
Ein ***Abhängigkeitsgraph*** *ist ein gerichteter Graph $G = (V, E)$, wobei V die Menge der Knoten (Elemente) ist und $E \subseteq V \times V$ die Menge der gerichteten Kanten darstellt. Eine Kante $(u, v) \in E$ zeigt an, dass das Element v vom Element u abhängig ist.*

Das Abhängigkeitsmaß ist eine Kennzahl, die ausdrückt, wie stark eine Aktivität A von einer anderen Aktivität B abhängig ist, basierend auf der beobachteten Reihenfolge in einem Ereignislog. Sie wird verwendet, um zu entscheiden, welche Kantengewichte dem Abhängigkeitsgraphen zugeordnet werden.

Definition 5 *Abhängigkeitsmaß*

$$a \Rightarrow_W b = \frac{|a >_W b| - |b >_W a|}{|a >_W b| + |b >_W a| + 1} \tag{2.6}$$

$$a \Rightarrow_W a = \frac{|a >_W a|}{|a >_W a| + 1} \tag{2.7}$$

- $|a >_W b|$*: Anzahl der Fälle, in denen a direkt vor b auftritt*
- $|b >_W a|$*: Anzahl der Fälle, in denen b direkt vor a auftritt*
- $|a >_W a|$*: Anzahl der Fälle, in denen a nach a auftritt*

Die im Nenner vorkommende +1 vermeidet die Division durch Null und stabilisiert die Metrik bei wenigen Aktivitäten.

(Weijters et al. 2006)

2.4 Process Discovery Algorithmen

Die Aufgabe von Process Discovery Algorithmen ist es, aus Ereignislogs Prozessmodelle zu erstellen. Hierfür gibt es verschiedene Algorithmen. Nach der Entwicklung des ersten Process Mining Algorithmus, dem α-Miner (Van der Aalst et al. 2004), wurden weitere Algorithmen entwickelt. Die relevantesten werden im Folgenden kurz erläutert.

α-Miner

Der α-Miner war der erste Algorithmus seiner Art, um aus einem Ereignislog Prozessmodelle zu erzeugen. Der Nachteil des α-Miners ist die Unfähigkeit, mit Rauschen umzugehen. Es wird außerdem nicht zwischen seltenen und häufig auftretenden Aktivitäten unterschieden. Sobald eine Aktivität auftritt, wird diese ins Prozessmodell aufgenommen, ungeachtet der relativen Auftretungshäufigkeit (Bogarín Vega et al. 2018, Mishra et al. 2018).

Heuristic-Miner

Im Gegensatz zum α-Miner kann der Heuristic-Miner mit Rauschen umgehen und beachtet bei der Erstellung des Prozessmodells auch relative Häufigkeiten von Aktivitäten. Ereignislogs mit selten auftretenden Aktivitäten und fehlerhaften Daten können vom Heuristic-Miner verarbeitet werden (Laue et al. 2020). Unstrukturierte Daten können mit diesem Miner verarbeitet werden, deshalb ist dieser für den vorliegenden Anwendungsfall als geeignet einzustufen (Mishra et al. 2018, Gupta 2014). Der Nachteil der vom Heuristic-Miner erstellten Prozessmodelle ist, dass diese nicht zwingend *sound* sein müssen. Es kann also zu *Deadlocks* im erstellten Prozessmodell kommen (Mishra et al. 2018).

Inductive-Miner

Der Inductive-Miner kann ähnlich dem Heuristic-Miner ebenfalls mit unstrukturierten Daten und Rauschen umgehen. Im Gegensatz zum Heuristic-Miner sind die vom Inductive-Miner erstellten Prozessmodelle auch immer *sound* (Leemans et al. 2013). Die Vorgehensweise des Inductive-Miners ist zuerst, einen Directly-Follows-Graphen

zu erstellen und diesen in verschiedene Bereiche zu partitionieren (Laue et al. 2020). Die einzelnen Bereiche sind folgendermaßen miteinander verknüpft: *Sequenz*,*AND*, *XOR* und *Schleife* (Leemans et al. 2013). Die resultierenden Modelle sind *sound* und verfügen über eine hohe *Fitness*, die bei Bedarf verringert werden kann, um die *Precision* des Modells zu erhöhen (Leemans et al. 2013, Laue et al. 2020).

Ist es gewünscht, einen Schwellwert für selten auftretendes Verhalten zu verwenden, so wird eine Variation des Inductive-Miners, der sogenannte *Inductive-Miner frequency-based* verwendet. Gewisse Folgerelationen von Aktivitäten werden dann nicht mehr Teil des erstellten Prozessmodells sein (Leemans et al. 2013). Für unstrukturierte Prozesse ist diese Eigenschaft von Vorteil, da seltene Ausführungspfade ausgelassen werden können. Es werden dann ausschließlich die häufigst auftretenden Prozessabläufe dargestellt.

Fuzzy-Miner

Der Fuzzy-Miner, vorgestellt von Günther and Van Der Aalst (2007) vereinfacht komplexe und unstrukturierte Prozessmodelle. Dies geschieht, indem wichtige Aktivitäten und Beziehungen anhand von Signifikanz und Korrelation identifiziert und hervorgehoben werden, während unwichtige Details entfernt oder zusammengefasst werden. Das Ziel des Miners ist es, anpassbare, verständliche Prozessdarstellungen zu erzeugen, die auf das Wesentliche fokussiert sind und verschiedene Detailstufen je nach Analysezweck ermöglichen.

Habit-Miner

Der Habit-Miner wurde von Dimaggio et al. (2016) mit dem Zweck entwickelt, menschliche Tagesabläufe zu erkennen. Der Miner basiert auf dem Fuzzy-Miner und versucht, die Schwierigkeit zu adressieren, dass ein Mensch, insbesondere im privaten Umfeld, keinem festen Prozessablauf folgt. Durch geschickte Datenvorverarbeitung versuchen die Autoren, aus Sensoraktivitäten in einem Smart Home einen Tagesablauf einer Hausbewohnerin zu identifizieren, so genannte *Habits*. Das resultierende Ereignislog wird nun vom Fuzzy-Miner weiterverarbeitet.

Fodina-Algorithmus

Der Fodina-Algorithmus ist laut den Entwicklern vanden Broucke and De Weerdt (2017), sich wiederholende Aktivitäten in Prozessabläufen korrekt einzuordnen. Sich wiederholende Aktivitäten treten in von Menschen ausgeführten Tagesabläufen häufig auf. Der Fodina-Algorithmus basiert auf dem Heuristic-Miner und adressiert einige Schwächen des Heuristic-Miners, wie beispielsweise das korrekte Identifizieren redundanter Aktivitäten. Das Problem der doppelten Aktivitäten in Prozessmodellen wird bei dem Fodina-Algorithmus mit Hilfe eines sogenannten *Task-Logs* adressiert, einer Abwandlung des Ereignislogs. Im Gegensatz zu Aktivitäten, die sich normalerweise in Ereignislogs befinden, beinhaltet ein *Task* im *Task-Log* einen Kontext, der nachfolgende und vorherige Aktivitäten mit einbezieht. Eine Aktivität kann also in mehreren *Tasks* enthalten sein. Bei der Erstellung eines Modells werden nun für alle *Tasks* die Anzahl der Folgebeziehungen gezählt. Anschließend wird ein Abhängigkeitsgraph erstellt und überprüft, ob dieser zusammenhängend ist. Nicht verbundene Abschnitte werden entfernt. Schlussendlich wird der Abhängigkeitsgraph in ein Causal-Net überführt. Die Evaluierung aus der Veröffentlichung von vanden Broucke and De Weerdt (2017) zeigt, dass doppelte Aktivitäten im resultierenden Causal-Net korrekt dargestellt werden.

2.5 Process Mining Software

Neben vielen kommerziellen Process Mining Werkzeugen gibt es einige Open-Source Projekte, wie beispielsweise *ProM Tools* (Van Dongen et al. 2005) oder Apromore (La Rosa et al. 2011). Diese sind jedoch mitunter fehlender Schnittstellen schwer in eigene Projekte integrierbar (Berti et al. 2019).

Process Mining for Python (PM4Py)

Das vom Fraunhofer-Institut für Angewandte Informationstechnik (FIT) entwickelte Process Mining for Python (PM4Py)[1] ist ein auf Python basierendes Framework, das sich insbesondere für Projekte, die in Python geschrieben sind, eignet.

[1] `https://processintelligence.solutions`

Durch das Verwenden von Python als Programmiersprache ist eine einfache Einbindung von der Vielzahl von Python-Bibliotheken möglich, was PM4Py zu einem mächtigen Werkzeug macht. Die PM4Py-Bibliothek verfügt über verschiedene Process Mining Algorithmen von *Process Discovery* bis *Conformance Checking*. Visualisierungsmöglichkeiten der resultierenden Prozessmodelle sind ebenso möglich. Aufgrund der oben genannten Vorteile und der freien Verfügbarkeit wird in dieser Arbeit mit der PM4Py-Bibliothek gearbeitet.

Fluxicon – Disco

Disco ist eine Process Mining Software von Fluxicon[2], welche entwickelt wurde, um Prozessanalysen benutzerfreundlich zu ermöglichen. Mithilfe von Disco können Prozessmodelle aus Ereignislogs erstellt werden. Die im Ereignislog aufgezeichneten Abläufe werden visualisiert und können mithilfe von Filtern interaktiv angepasst werden. Zur Prozessentdeckung verwendet Disco eine Weiterentwicklung des „Fuzzy Miners“. Komplexe Prozessdaten sollen mit diesem Miner auch zuverlässig abbildbar sein. Die hohe Nutzerfreundlichkeit ermöglicht eine Benutzung der Software ohne tiefergehende Kenntnisse in Process Mining. (Günther and Rozinat 2012)

ProM (Process Mining Framework)

ProM (Process Mining Framework)[3] ist ein Tool, welches entwickelt wurde, um Probleme der Kompatibilität bestehender Tools zu lösen. Das Tool ist Open Source, erweiterbar und plattformübergreifend. Im Zentrum der Software stehen Ereignislogs, die mithilfe einer großen Vielfalt an Plug-Ins analysiert werden können. Aus den Ereignislogs lassen sich Prozessverläufe, Ressourcen und Fallmerkmale extrahieren. Die Ergebnisse können unter anderem als Petri-Netze, soziale Netzwerke visualisiert werden und gegebenenfalls mithilfe von weiteren ProM-Plug-Ins weiter analysiert werden. Speziell im Bereich der Forschung war und ist ProM eine wichtige Softwarelösung (Van Dongen et al. 2005).

[2] `https://fluxicon.com/disco/`
[3] `https://promtools.org`

2.6 Sensorumgebungen

Smart Home

Smart Homes sind Wohnungsumgebungen, in denen unterschiedliche Komponenten des Wohnraums mithilfe von IT-Infrastruktur miteinander verbunden sind. Die Motivation, einen Wohnraum mithilfe von Sensoren und IT-Infrastruktur auszustatten, kann vielfältig sein. So kann mithilfe von Automatisierungen ein Zugewinn an Komfort entstehen. Durch die automatisierte und optimierte Steuerung von Geräten ist es auch möglich, Energie zu sparen, indem Lasten an die Verfügbarkeiten von regenerativen Energien angepasst werden (Schiefer 2015).

Im Kontext der zunehmenden Vernetzung von Geräten im Smart Home ist insbesondere der Zugriff auf das Heimnetzwerk von außen zu berücksichtigen. Dabei stellen sowohl die Art der eingesetzten Sensoren als auch die gewährten Zugriffsberechtigungen zentrale Faktoren dar, die maßgeblich die Sicherheit und den Schutz der Privatsphäre der Bewohner beeinflussen können. Insbesondere bei sensiblen Sensordaten besteht das Risiko einer erheblichen Verletzung der Privatsphäre, wenn unbefugte Dritte Zugang zu diesen Informationen erhalten (Pohlmann 2021).

Smart Factory

Eine Smart Factory ist eine Produktionsumgebung, die darauf ausgelegt ist, sich dynamisch und schnell auf sich ändernde Rahmenbedingungen anzupassen. Insbesondere bei zunehmend komplexen Marktbedingungen ist eine flexible und adaptive Produktionsumgebung sinnvoll. Eine Smart Factory basiert auf einem sogenannten cyber-physischen Produktionssystem (CPPS), welches die Komponenten der physischen Welt mit digitalen Elementen verbindet. Dies ermöglicht es, Produktionssysteme in (nahezu) Echzeit zu steuern (Hozdić 2015).

2.7 Clustering

Das Ziel des Clusterings ist es, ähnliche Elemente zu finden und in Gruppen, sogenannte Cluster, zusammenzufassen. Die für diese Arbeit relevanten Methoden werden im Folgenden erläutert.

k-Means

Der k-Means Clustering-Algorithmus teilt eine Menge an n-dimensionalen Datenpunkten in k Cluster auf, indem die quadratischen Abstände der k-Cluster-Zentren (Schwerpunkte bzw. Centroiden) minimiert werden (Hartigan and Wong 1979). Die folgenden Schritte beschreiben den k-Means-Algorithmus grob (Jain 2010):

1. Initial-Aufteilung der Daten mit k-Cluster wählen (meist zufällig)
2. Neue Aufteilung der Cluster berechnen durch Zuteilung der Punkte zu den jeweils am nächsten befindlichen Cluster-Zentren
3. Neue Cluster-Zentren berechnen
4. Schritte 3 und 4 so lange wiederholen, bis es zu keiner großen Veränderung mehr kommt

Die oben genannten Kernelemente des k-Means-Verfahrens sind im Folgenden als Pseudocode in Algorithmus 1 umgeschrieben. Laut (Alpaydin 2020) gibt es zwei

Algorithmus 1 k-Means Algorithmus (Alpaydin 2020)

function K-MEANS
    Initialisiere Cluster-Zentren μ_j, mit $j = 1, \dots k$ zufällig
    do
        for $i = 1$ to m **do**
            Berechne $j' = argmin_{j=1,\dots,k} d(x_i, \mu_j)$
            $r_{ij'} = 1$, $r_{ij} = 0$ für alle $j' \neq j$
        end for
        for $j = 1$ to k **do**
            Berechne $\mu_j = \frac{\sum_i r_{ij} x_i}{\sum_i r_{ij}}$
        end for
    while Änderung der Clusterzuordnung r_{ij} zu voriger Iteration
    return $\{\mu_1, \dots, \mu_k\}$, r_{ij}
end function

erwähnenswerte Schwächen des k-Means Algorithmus. Zum einen ist das Ergebnis des Clusterings stark von der initialen Wahl der Clusterzentren μ abhängig. Abhilfe schafft unter anderem die Methode, anstelle zufälliger Werte zufällige Datenpunkte aus der zu clusternden Datenmenge als Zentren auszuwählen. Zum anderen werden die geclusterten Datenpunkte nur genau einem Cluster zugeordnet. Dies lässt sich

durch Erweiterungen des k-Means Algorithmus verbessern, zum Beispiel mit dem *Fuzzy C-Means Clustering* (Bezdek et al. 1984).

k-Medoids

Der k-Medoids-Algorithmus ist eine Variante des k-Means-Clusteringverfahrens. Beim k-Means-Algorithmus werden die Clusterzentren (Centroids) als Mittelwerte der jeweiligen Cluster berechnet und sind somit nicht unbedingt tatsächlich auftretende Datenpunkte. Die Centroiden des k-Medoids-Algorithmus sind jedoch immer ein Datenpunkt aus den zu clusternden Objekten des Datensatzes. Die Clusterzentren sind definiert als ein Punkt innerhalb eines Clusters, deren durchschnittliche Unähnlichkeit zu allen anderen Punkten im selben Cluster minimal ist (Kaufman and Rousseeuw 2009).

Self-Organising Map

Die Self-Organising Map (zu deutsch *Selbstorganisierende Karte*, auch bekannt als Kohonen-Netz), entwickelt von *Teuvo Kohonen*, ist ein unüberwachtes maschinelles Lernverfahren. Bei der Self-Organising Map handelt es sich um ein künstliches neuronales Netz, das dazu geeignet ist, Muster zu erkennen und für die Verarbeitung semantischer Informationen (Kohonen 1990). Mithilfe einer SOM können mehrdimensionale Daten auf einer niedrigeren Dimension dargestellt werden. Hieraus ergibt sich ein großer Vorteil der SOM, denn durch die Umwandlung von mehrdimensionalen Daten zu zwei oder drei Dimensionen können höherdimensionale Daten gut grafisch dargestellt werden. Die Ursprungsdaten werden komprimiert, behalten jedoch die topologischen und metrischen Eigenschaften bei (Kohonen 1990).

Ausgangslage ist ein n-dimensionaler Vektor der Form:

$$x = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^n \tag{2.8}$$

Dieser soll nun auf einem beispielsweise zweidimensionalen Netz mit I Neuronen übertragen werden. Der Gewichtsvektor des Neuronens i lautet:

$$m_i = [m_{i1}, m_{i2}, \ldots, m_{in}]^T \in \mathbb{R}^n \tag{2.9}$$

Der Input-Vektor ist mit jedem Neuron des Netzes miteinander über den Gewichtsvektor m_i verbunden. Um beim Training des Netzes feststellen zu können, welches Neuron zu dem Input-Vektor passt, wird der Input-Vektor x mit jedem Gewichtsvektor verglichen. Der Gewichtsvektor m_i mit der geringsten euklidischen Distanz zum Input-Vektor ist der „Gewinner“ (BMU) m_c.

$$\|x - m_c\| = \min_i \{\|x - m_i\|\} \tag{2.10}$$

Zum Lernprozess dieses neuronalen Netzes gehört nun die Anpassung der Gewichte. Um die topologische Ordnung zu erhalten, wird nicht nur die BMU angepasst, sondern auch benachbarte Neuronen aus der Nachbarschaft N_c des Gewinner-Neurons c. Neuronen außerhalb der Nachbarschaft N_c bleiben bei der Anpassung unberührt. Die Größe der Nachbarschaft kann als Funktion der Zeit verkleinert werden, um so bessere Ergebnisse zu erzielen. Der Aktualisierungsprozess kann folgendermaßen beschrieben werden:

$$m_i(t+1)(x) = \begin{cases} m_i(t) + \alpha(t)[x(t) - m_i(t)] & i \in N_c(t), \\ m_i(t) & i \notin N_c(t) \end{cases} \tag{2.11}$$

$\alpha(t)$ ist der zeitabhängige Gewichtsfaktor der Nachbarschaftsgröße und sollte über die Zeit kleiner werden. Es gilt $0 < \alpha(t) < 1$. In Abbildung 2.5 sind die Neuronen des Kohonennetzes als Kreise dargestellt und die drei verschiedenen Nachbarschaften $N_c(t_1)$, $N_c(t_2)$, $N_c(t_3)$ als Hexagone eingezeichnet (Kohonen 1990).

Bei großen Kohonennetzen kann es für eine weitere quantitative Analyse notwendig sein, dass ähnliche Neuronen mithilfe eines zweiten Clustering-Algorithmus erneut geclustert werden. Dies ist in Abbildung 2.6 grafisch dargestellt.

Für die zweite Clustering-Stufe eignet sich unter anderem der in Abschnitt k-Means vorgestellte k-Means-Algorithmus. Der größte Vorteil dieser zweistufigen Vorgehensweise ist der reduzierte Rechenaufwand im Gegensatz zu einem einstufigen Vorgehen (Vesanto and Alhoniemi 2000).

2.8 Distanzfunktionen

Um Traces in aussagekräftige Cluster zu gruppieren, ist es notwendig, eine Kennzahl zur Ähnlichkeit der Traces zu finden.

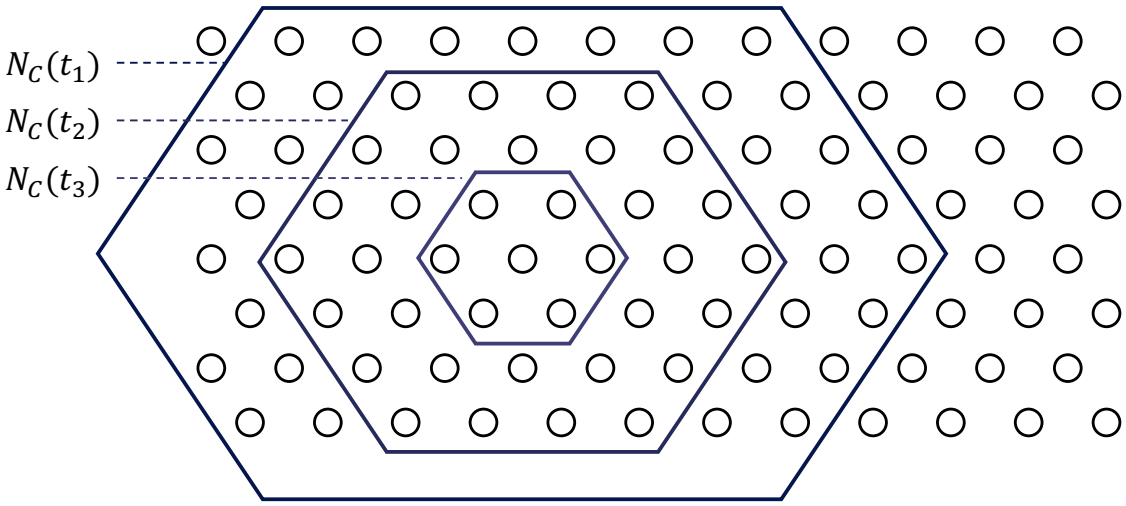


Abbildung 2.5: Zeitlicher Verlauf der Nachbarschaftsgröße einer SOM (Kohonen 1990)

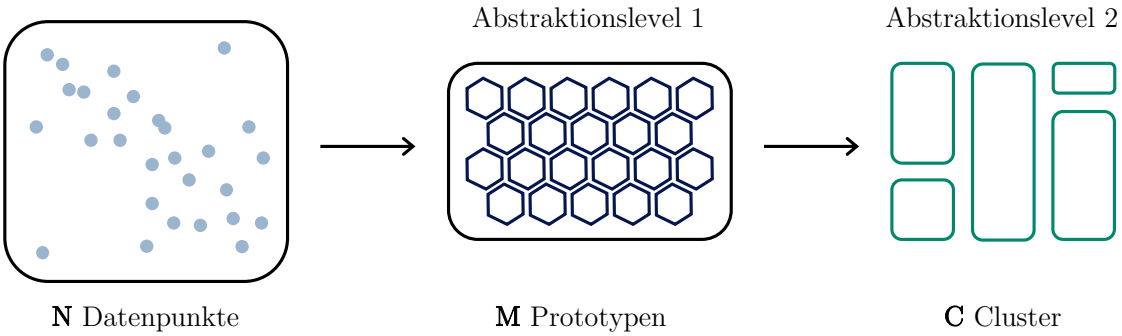


Abbildung 2.6: SOM Clustering (nach Vesanto and Alhoniemi (2000))

Euklidische Distanz

Der euklidische Abstand zwischen zwei Vektoren x und y, deren Länge n, lässt sich mit der Formel 2.12 berechnen. $d(x, y)$ steht für die Distanz zwischen den beiden Vektoren x und y. n entspricht der Länge der Vektoren x und y ($n = |x| = |y|$). i ist die Laufvariable, die mithilfe der Summe ($\Sigma_{i=1}^{n}$) über jedes Element der Vektoren x und y iteriert.

$$d(x, y) = \|x - y\|_2 = \sqrt{\Sigma_{i=1}^{n}(x_i - y_i)^2} \tag{2.12}$$

Das Problem beim Verwenden der euklidischen Distanz ist, dass die Aussagekraft des Distanzmaßes über die Ähnlichkeit der Vektoren davon abhängig ist, wie die Sensoren in den Vektoren repräsentiert sind. Enthalten die Vektoren wie in Kapitel 2.1 willkürliche Sensorbezeichnungen (z.B. Sensor A, Sensor B, Sensor C, ...), hat der Distanzwert, der mithilfe der euklidischen Distanz berechnet wurde, zwischen den Vektoren kaum eine semantische Bedeutung. Um die euklidische Distanz dennoch sinnvoll anwenden zu können, kann entweder der Vektor transformiert werden (siehe Kapitel 3.3) oder eine alternative Distanzberechnung verwendet werden.

2.9 Hyperparameteroptimierung

Bei größeren Optimierungsproblemen, wie dem hier vorliegenden, gibt es oftmals eine Vielzahl an Parametern. Die korrekte Kombination der Parameter ist entscheidend für die Qualität des resultierenden Prozessmodells (Bergstra et al. 2015). Je größer die Menge an Parametern und je größer der Wertebereich dieser Parameter ist, desto größer wird der zu untersuchende Suchraum. Das Finden der besten Parameterkombinationen kann bei einem großen Suchraum sehr lange dauern. Um die optimale Lösung zu finden, gibt es verschiedene Strategien, die in den folgenden Unterkapiteln erläutert werden.

Neben der vollständigen Erkundung des Suchraums gibt es die Möglichkeit, mithilfe von statistischen Methoden die Suche zu beschleunigen. Um die Anzahl an zu berechnenden Parameterkombinationen zu verringern, versucht das verwendete Verfahren, Parameterkombinationen geschickt zu wählen. Mithilfe eines zu berechnenden Zielfunktionswertes wird die Wahl vergangener Parameterkombinationen mit einbezogen und versucht, zugrunde liegende Muster in den Parameterkombinationen zu erkennen, um bei der Parameterwahl der nächsten Iteration einen besseren Zielfunktionswert zu erreichen.

2.10 Aktivitätserkennung

Es gibt bereits eine Vielzahl von Forschungsarbeiten, die sich mit dem Thema Aktivitätserkennung von Ereignissen und Aktivitäten befassen. Im Folgenden werden verwandte Ansätze betrachtet, die Sensordaten nutzen, um sie in diskrete Ereignisdaten auf höherer Abstraktionsebene zu übersetzen oder Ansätze, die Process

Mining auf rohe Sensordaten anwenden. Der Fokus liegt auf Smart Homes, da Daten von Smart Home-Sensoren für die Evaluierung verwendet werden.

Die Aktivitätserkennung in Smart Homes ist weit verbreitet und basiert auf verschiedenen Sensortypen wie Bewegungssensoren oder Videodaten (Nentwig and Stamminger 2010, Zhang and Sawchuk 2012, Diete et al. 2018). Verbreitet ist ebenfalls die Analyse von Daten von Wearables (Sztyler et al. 2016, Larue et al. 2015). In jüngster Zeit werden zunehmend Deep Learning-Methoden zur Erkennung und Vorhersage von Aktivitäten in IoT-Umgebungen eingesetzt (De Weerdt et al. 2013). Im Gegensatz zu klassischen Ansätzen leiten Deep Learning-Netzwerke automatisch Merkmale aus den Daten ab und liefern vielversprechende Ergebnisse in verschiedenen Bereichen. Insbesondere im Bereich Smart Home oder *Ambient Assisted Living* gibt es erste Ansätze, die Aktivitäten auf Basis von Sensorereignisdaten erkennen (Choi et al. 2013, Gallicchio et al. 2017, Wang et al. 2016, 2019). Auch diese Art von Ansätzen identifiziert einfache Aktivitäten, wie zum Beispiel Laufen und Treppe steigen (Dernbach et al. 2012, Sztyler 2019). Komplexe Aktivitäten wie die täglichen Aktivitäten von Menschen können nur mit zusätzlichen Sensoren identifiziert werden (Dernbach et al. 2012, Nguyen et al. 2019). Die vorgestellte Methode zur Erkennung von Prozessmodellen aus rohen Sensordaten erfordert ebenfalls eine manuelle Kennzeichnung der finalen Aktivitätscluster. Jedoch bedeutet die Prozessmodellansicht auf die Daten eine Verbesserung im Hinblick auf die Bewertung der Datenaggregationsqualität, wozu Deep Learning-Ansätze nicht unbedingt in der Lage sind.

Die Abbildung von Low-Level-Ereignissen auf Aktivitäten für das Process Mining ist immer noch eine Herausforderung (Koschmider et al. 2019). Leotta et al. (2015) beabsichtigen in weiteren Forschungsarbeiten, ähnliche Techniken wie die hier vorgestellten zu verwenden: Aktuell werden jedoch nur die Herausforderungen diskutiert. Der aktuelle Status ist, dass Ansätze nur Wahrscheinlichkeiten von Abbildungen angeben, da es oft mehr als eine mögliche Lösung gibt (Van Der Aa et al. 2017). Der hier vorgestellte Ansatz zur Aggregation von Ereignissen, in Kombination mit unüberwachten Lernverfahren, soll diese Lücke schließen.

Verwandte Literatur zur Aktivitätserkennung für Process Mining verwendet entweder überwachte Lernverfahren (Tax et al. 2018b, Sztyler et al. 2016) oder visualisiert menschliche Gewohnheiten bloß (Leotta et al. 2020), um Aktivitäten genau zu identifizieren. Es gibt einige Arbeiten, die Aktivitäten aus High-Level-Ereignissen durch

unüberwachte Lernverfahren erkennen (Mannhardt and Tax 2017, Tax et al. 2018b, Alharbi et al. 2018), die in dieser Arbeit verglichen wurden. Diese verwandten Arbeiten von Mannhardt and Tax (2017), Tax et al. (2018b) und Alharbi et al. (2018) verwenden Muster oder lokale Prozessmodelle, um Ereignisdaten auf höheren Abstraktionsebenen zu aggregieren. Sie erlaubten es jedoch nicht, sinnvolle Aktivitäten aus unstrukturierten Daten zu entdecken. Für ungelabelte Trainingsdatensätze schlagen verwandte Ansätze vor, eine zeitbasierte Label-Verfeinerung (Tax et al. 2019) oder Standorte (Brzychczy and Trzcionkowska 2019) als Merkmale zu verwenden, um das Ereignislog zu segmentieren und daraus Aktivitäten zu abstrahieren. Allerdings erwarten die Methoden bereits bestimmte Repräsentationen von Traces.

3 Framework

Das folgende Framework nutzt als Eingabe ungelabelte Sensor-Rohdaten. Anschließend werden mithilfe eines unüberwachten Lernverfahrens die Daten aggregiert und geclustert, um Aktivitäten anhand von Clustern und sinnvollen Ereignisabfolgen zu identifizieren. Zur besseren Erkennung der Aktivitäten wird für jedes Cluster ein Prozessmodell ermittelt, welches die Aktivierungsreihenfolge der involvierten Sensoren dieses Clusters beinhaltet. Dies gibt einem Domänenexperten die Möglichkeit, sinnvolle Aktivitätsnamen für die Cluster zu finden. Die Aktivitäten dienen anschließend als Input für die auf Process Mining basierende Erkennung eines Modells, welches es ermöglicht, parallel stattfindendes und sich überschneidendes Verhalten in Sensorereignisdaten zu identifizieren.

Mithilfe von Process Mining werden Prozessmodelle aus den aggregierten Ereignisdaten entdeckt. Durch die Integration einer Feedback-Schleife kann festgestellt werden, wie empfindlich das entdeckte Prozessmodell auf Parameteränderungen, im Bezug auf eine vorgegebene Zielfunktionsgröße, reagiert. Dies schließt Parameteränderungen aller Verarbeitungsschritte des Frameworks ein.

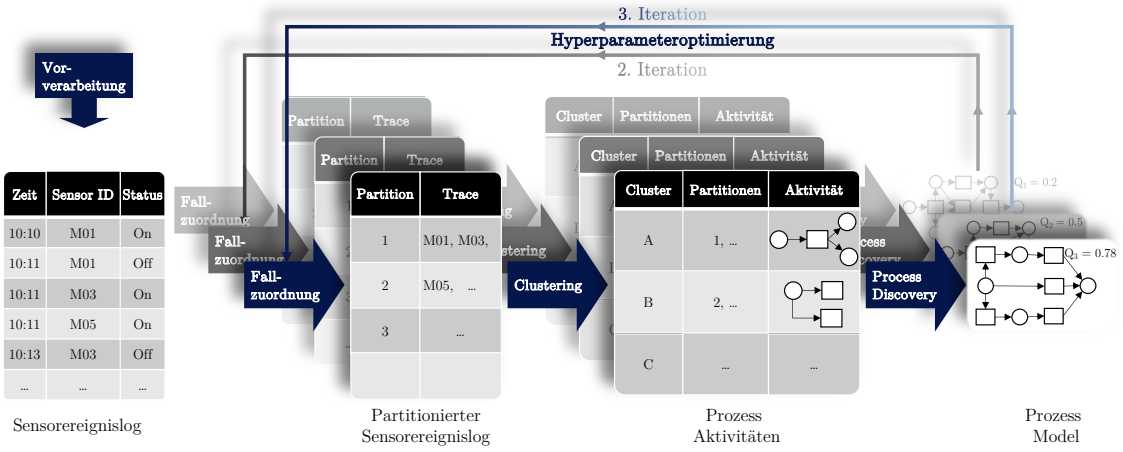


Abbildung 3.1: Framework (nach Janssen et al. (2026))

Das Framework gliedert sich in verschiedene Module, die grundsätzlich unabhängig voneinander sind und dementsprechend individuell verändert und erweitert werden können. Wie in Kapitel 4 beschrieben, steht zu Beginn der Auswertung ein ungelabelter Datensatz, der als Mindestanforderung Zeitstempel, Sensor-ID und Sensorstatus enthalten muss. Abbildung 3.1 zeigt das Framework, welches aus fünf Schritten besteht:

Vorverarbeitung: Für die Prozesserkennung müssen zunächst Vorverarbeitungsschritte durchgeführt werden, um eventuell vorhandenes Rauschen im Ereignislog zu filtern. Sofern es sich um einen Datensatz handelt, in dem eine einzige Entität beobachtet wird, werden Sensoraktivierungen entfernt, die nicht durch eine einzige Entität erklärt werden können. Für die räumliche Modellierung der ortsfesten Sensoren wird eine Adjazenzmatrix erstellt, die Nachbarschaftsbeziehungen und Distanzen abbildet. Dies erfolgt unter dem Einbezug von Hindernissen wie Wänden. Existiert keine Distanzmatrix, wird diese mithilfe des Floyd-Warshall-Algorithmus aus der Adjazenzmatrix berechnet. Dies stellt sicher, dass immer die kürzesten Wege zwischen allen Sensorpaaren berechnet werden. Um Lücken in der Sensorabdeckung zu berücksichtigen, wird ein hypothetischer Hilfssensor eingeführt, der eine konstante Distanz zu allen realen Sensoren hat.

Fallzuordnung: Bevor die eigentliche Prozesserkennung stattfinden kann, müssen die Sensordaten noch zwei weitere Vorverarbeitungsschritte durchlaufen: die Fallzuordnung und die Partitionierung der Fälle. Bei der Fallzuordnung geht es darum, Sensoraktivierungen korrekt einzelnen Entitäten zuzuordnen. Problematisch ist dies insbesondere bei Bewegungssensoren, bei denen keine eindeutige Entitätszuordnung stattfindet. Bei eindeutig identifizierbaren Entitäten können Sensorereignisse direkt und eindeutig zugeordnet werden; bei nicht eindeutig unterscheidbaren Entitäten ist die Anwendung von Heuristiken wie einer gewichteten Distanzfunktion notwendig. Zusätzlich müssen die Sensordaten sinnvoll partitioniert werden, um unterscheidbare Fälle zu bilden. Hierbei können entweder die Anzahl der Sensoraktivierungen, Aktivierungsdauern oder räumliche Begrenzungen als Kriterien dienen.

Clustering: Die zuvor partitionierten Sensorereignislogs werden zu Clustern zusammengefasst. Jedes Cluster repräsentiert idealerweise genau eine Aktivität. Dazu werden zunächst die Fälle zu Vektoren transformiert; dies ist notwendig,

um geeignete Distanzmaße für Clustering-Algorithmen anwenden zu können. Variable Längen der Fälle erschweren das Clustering. Zusätzlich ist es notwendig, aussagekräftige Distanzmaße zwischen Sensoraktivierungen festzulegen. Das vorgestellte Framework unterstützt drei Arten der Vektorisierung: Anzahl, Dauer und die Kombination von Anzahl und Dauer. Bei dieser Art von Transformation gehen jedoch die Richtungsinformationen der Bewegungen verloren. Anschließend erfolgt das Clustering ähnlicher Fälle mittels k-Means, k-Medoids oder Self-Organising Maps (SOM). Die resultierenden Cluster erhalten zunächst temporäre Pseudo-Labels. Diese werden nach Abschluss einer Qualitätsbewertung und Hyperparameteroptimierung manuell durch semantisch sinnvolle Aktivitätslabels ersetzt.

Process Discovery: Der Process Discovery Schritt besteht aus zwei Schritten: Zunächst werden aus den gebildeten Clustern Prozessmodelle in Form von Directly-Follows-Graphen (DFG) erstellt. Seltene Sensoraktivierungen werden zur Vereinfachung der Modelle herausgefiltert. Anschließend werden die Cluster mit provisorischen Aktivitätsbezeichnungen benannt. Erst zu einem späteren Zeitpunkt werden die Bezeichnungen der Aktivitäten durch einen Domänenexperten validiert bzw. interpretiert. Im zweiten Teil des Process Discovery wird versucht, aus den Sensordaten vollständige Tagesabläufe zu modellieren. Es wird dabei versucht, die Tage in verschiedene Tages- oder Wochenabschnitte (z.B. Morgen-, Abend- oder Wochenroutinen) zu unterteilen, um realistischere und besser interpretierbare Prozessmodelle zu erhalten. Diese Modelle werden mithilfe des Heuristic Miner und Inductive Miner erstellt und deren Qualität mittels Fitness-, Precision- und F_1-Scores bewertet.

Hyperparameteroptimierung: Die Hyperparameteroptimierung erfolgt mithilfe einer Feedbackschleife, welche die Parameterkombinationen iterativ anpasst. Diese soll dafür sorgen, die Qualität des generierten Prozessmodells zu maximieren. Die Qualität wird gemessen am tokenbasierten F_1-Score, dem harmonischen Mittel aus Precision und Fitness. Dazu wird der Tree of Parzen Estimators (TPE) verwendet, basierend auf der Hyperparametersuche nach Bergstra et al. (2013). Um verschiedene Zeitsegmente (z. B. Tages- oder Wochenabschnitte) in individuellen Prozessmodellen (sogenannten *Routinen*) angemessen zu berücksichtigen, wird ein gewichteter F_1-Score berechnet. Parameterwerte werden nicht fest vorgegeben. Die Wertebereiche der Parameter

können je nach Anwendungsdomäne durch einen Domänenexperten bestimmt werden. Dies soll die Effizienz und Übertragbarkeit des Ansatzes auf unterschiedliche Domänen sicherstellen.

3.1 Vorverarbeitung

Bevor mit der eigentlichen Prozesserkennung begonnen werden kann, müssen die im Folgenden beschriebenen Vorverarbeitungsschritte durchgeführt werden.

Diese Schritte filtern Rauschen im Ereignislog. Sofern sich mehrere Entitäten in einer der beobachteten Umgebungen aufhalten, werden den einzelnen Sensoraktivierungen Entitäts-IDs zugeordnet, um im späteren Auswertungsverlauf zwischen den jeweiligen Entitäten unterscheiden zu können. In den Datensätzen finden sich üblicherweise Bewegungsdaten zu mehreren Personen. Sind mehrere Sensoren aktiviert, kann daraus also nicht automatisch geschlossen werden, ob eine zweite Person anwesend ist. Im Falle einer solchen Konstellation gibt es zwei Vorgehensweisen. Die Vorgehensweise unterscheidet sich, ob eine eindeutige Identifizierung der Entitäten möglich ist oder nicht. Die unterschiedliche Vorgehensweise wird im Folgenden zusammengefasst.

Adjazenzmatrix

Sofern es sich bei den Sensoren im Datensatz um stationäre Sensoren handelt, ist es für die weiteren Verarbeitungsschritte wichtig, die Beziehungen, in denen die Sensoren relativ zueinander stehen, abzubilden.

Liegt eine Distanzmatrix (eine symmetrische $n \times n$ Matrix, die von jedem Sensor die kürzeste Distanz zu allen anderen Sensoren angibt) vor, kann diese verwendet werden.

Alternativ kann aus dem Lageplan der Sensoren eine Adjazenzmatrix erstellt werden. Die Adjazenzmatrix gibt die Entfernung der direkten Nachbarschaft von Sensoren an (siehe Abbildung 3.2). Die symmetrische $n \times n$ Matrix beinhaltet für benachbarte Sensorpaare (i, j) die jeweilige Distanz als Element. Nicht benachbarte Sensorpaare und der Sensor zu sich selbst (i, i) enthalten eine 0 (Knauer and Knauer 2019).

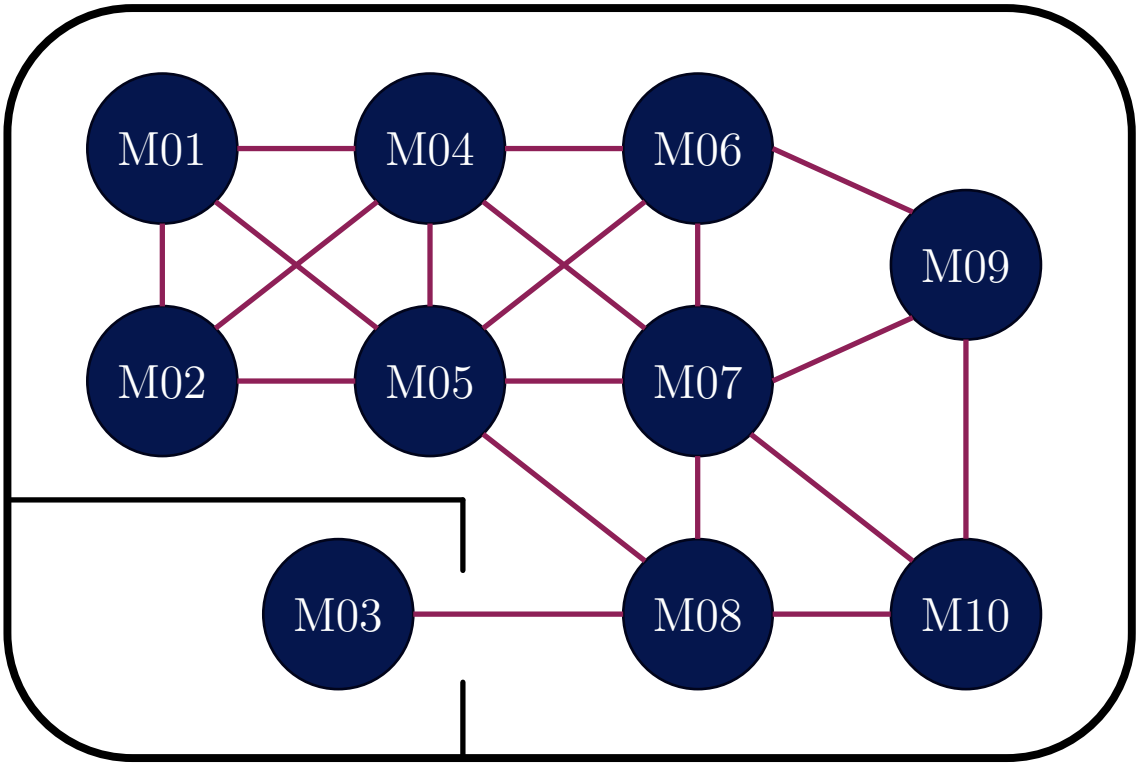


Abbildung 3.2: Sensoren als Graph

Hierbei ist zu unterscheiden, ob sich die Sensoren als Graph mit oder ohne Kantengewichten darstellen lassen:

Ohne Kantengewichte: Sollten im Datensatz keine expliziten Distanzwerte zur Verfügung stehen, wird von einer konstanten Distanz zwischen allen benachbarten Sensoren ausgegangen. In diesem Fall wird für jedes Nachbarschaftspaar (i, j) der gleiche Distanzwert (standardmäßig eine Eins) in der Adjazenzmatrix eingesetzt. Je nach Lageplan der Sensoren kann die Annahme der Äquidistanz der Sensoren untereinander eine mögliche Annahme und damit eine ausreichend gute Näherung sein.

Mit Kantengewichten: Liegt für den Lageplan ein Maßstab vor, oder lässt sich auf andere Art und Weise abschätzen, wie weit die jeweiligen Sensoren voneinander entfernt liegen, können diese Werte in die Adjazenzmatrix eingetragen werden.

Bei der Erstellung der Adjazenzmatrix ist darauf zu achten, Wände oder andere Hindernisse in die Nachbarschaftsbeziehung mit einzubeziehen. In Abbildung 3.2 sind die Sensoren `M02` und `M03` zwar nahe beieinander, jedoch durch eine Wand

	M01	M02	M03	M04	M05	M06	M07	M08	M09	M10
M01	0	1	0	1	1	0	0	0	0	0
M02	1	0	0	1	0	0	0	0	0	0
M03	0	0	0	0	0	0	0	1	0	0
M04	0	0	0	0	0	0	0	1	0	0
M05	1	1	0	1	0	1	1	1	0	0
M06	0	0	0	1	1	0	1	0	1	0
M07	0	0	0	1	1	1	0	1	1	1
M08	0	0	1	0	1	0	1	0	0	1
M09	0	0	0	0	0	1	1	0	0	1
M10	0	0	0	0	0	0	1	1	1	0

Tabelle 3.1: Adjazenzmatrix zu Abbildung 3.2

getrennt. Sie werden also nicht als direkte Nachbarn in der Adjazenzmatrix (Tabelle 3.1) vermerkt.

Distanzmatrix

Aus der Adjazenzmatrix 3.1 wird mithilfe des *Algorithmus von Floyd und Warshall* (Floyd 1962) eine Distanzmatrix erstellt. Der zugehörige Pseudocode ist in Algorithmus 2 dargestellt.

Um die Distanzmatrix zu erstellen, wird zuerst die Adjazenzmatrix geklont (Algorithmus 2, Zeile 2). Anschließend werden in den beiden verschachtelten *for*-Schleifen ab Zeile 3 für jedes Knotenpaar der Distanzmatrix, welches keine direkte Verbindung zueinander hat, also einen Wert von 0 in der Adjazenzmatrix hat, auf $+\infty$ gesetzt. Knotenpaare zu sich selbst bekommen einen Distanzwert von 0 zugeordnet. Durch das Klonen der Adjazenzmatrix in Zeile 2 hat die Zwischenergebnismatrix zusätzlich zu den $+\infty$ und 0 Einträgen bei allen Elementen, die direkte Nachbarn im Graphen sind, den korrespondierenden Distanzwert. Ab Zeile 13 wird für jede Knotenkombination überprüft, ob das Hinzufügen der Kanten ik und kj zu einem kürzeren

Algorithmus 2 Floyd-Warshall Algorithmus (Floyd 1962)

```
1: function FloydWarshall(adjacency_matrix)
2:     distance_matrix ← adjacency_matrix
3:     for n = 0 to len(distance_matrix) do
4:         for m = 0 to len(distance_matrix) do
5:             if distance_matrix[n][m] = 0 then:
6:                 distance_matrix[n][m] = +∞
7:             end if
8:             if n = m then:
9:                 distance_matrix[n][m] = 0
10:            end if
11:        end for
12:    end for
13:    for k = 0 to len(distance_matrix) do
14:        for i = 0 to len(distance_matrix) do
15:            for j = 0 to len(distance_matrix) do
16:                distance_matrix[i][j] =
                       min{distance_matrix[i][j],
                           distance_matrix[i][k] + distance_matrix[k][j]}
17:            end for
18:        end for
19:    end for
20:    return distance_matrix
21: end function
```

Weg als die direkte Verbindung ij führt. Die am Ende des Algorithmus ausgegebene Distanzmatrix beinhaltet für jedes Sensorpaar die kürzeste Entfernung. Die resultierende Distanzmatrix ist in Tabelle 3.2 zu sehen.

Es kann in Anwendungsszenarien der realen Welt vorkommen, dass aufgrund von toten Winkeln zu bestimmten Zeiten kein einziger Sensor aktiviert ist. Um mit diesen Abdeckungslücken umgehen zu können, wird ein hypothetischer Hilfssensor $\mathtt{M}_0$ eingeführt, der von jedem Sensor eine feste Entfernungseinheit entfernt ist. Die Entfernungseinheit zum hypothetischen Hilfssensor $\mathtt{M}_0$ kann je nach Anwendungsdomäne angepasst werden.

	M01	M02	M03	M04	M05	M06	M07	M08	M09	M10
M01	0	1	3	1	1	2	2	2	3	3
M02	1	0	3	1	2	3	3	2	4	3
M03	3	3	0	3	2	3	2	1	3	2
M04	3	3	2	0	2	3	2	1	3	2
M05	1	1	2	1	0	1	1	1	2	2
M06	2	2	3	1	1	0	1	2	1	2
M07	2	2	2	1	1	1	0	1	1	1
M08	2	2	1	2	1	2	1	0	2	1
M09	3	3	3	2	2	1	1	2	0	1
M10	3	3	2	2	2	2	1	1	1	0

Tabelle 3.2: Distanzmatrix zu Abbildung 3.2

3.2 Fallzuordnung

Ohne Vorverarbeitung ist es nicht möglich, aus dem unstrukturierten Sensorereignislog Fälle bzw. Traces zu finden, die sich direkt von Process Mining Verfahren verarbeiten lassen. Es ergeben sich zwei Herausforderungen:

Zuordnung zu Fällen: Bei der Zuordnung zu Fällen werden Sensoraktivierungen herausgefiltert, die als Rauschen identifiziert werden können. Eine zusätzliche Schwierigkeit besteht darin, mehrere Entitäten auseinanderzuhalten und die Sensorwerte korrekten Fällen zuzuordnen. Befinden sich in einem Bereich mehrere Entitäten, muss bei jedem Sensorereignis entschieden werden, welcher Entität dieses zugeordnet werden soll. Dies ist herausfordernd, wenn die Art des Sensors keine Zuordnung zu einer bestimmten Entität zulässt, wie beispielsweise bei Bewegungsmeldern.

Partitionierung der Fälle: Bei einer kontinuierlichen Aufzeichnung der Sensorwerte ist es schwierig zu erkennen, wann eine Aktivität beginnt und wann sie endet. Meist werden keine zusätzlichen Informationen im Sensorereignislog vermerkt, die einen direkten Rückschluss auf den Start und das Ende von Aktivitäten zulassen. Für die weiteren Auswertungsschritte ist es jedoch wichtig,

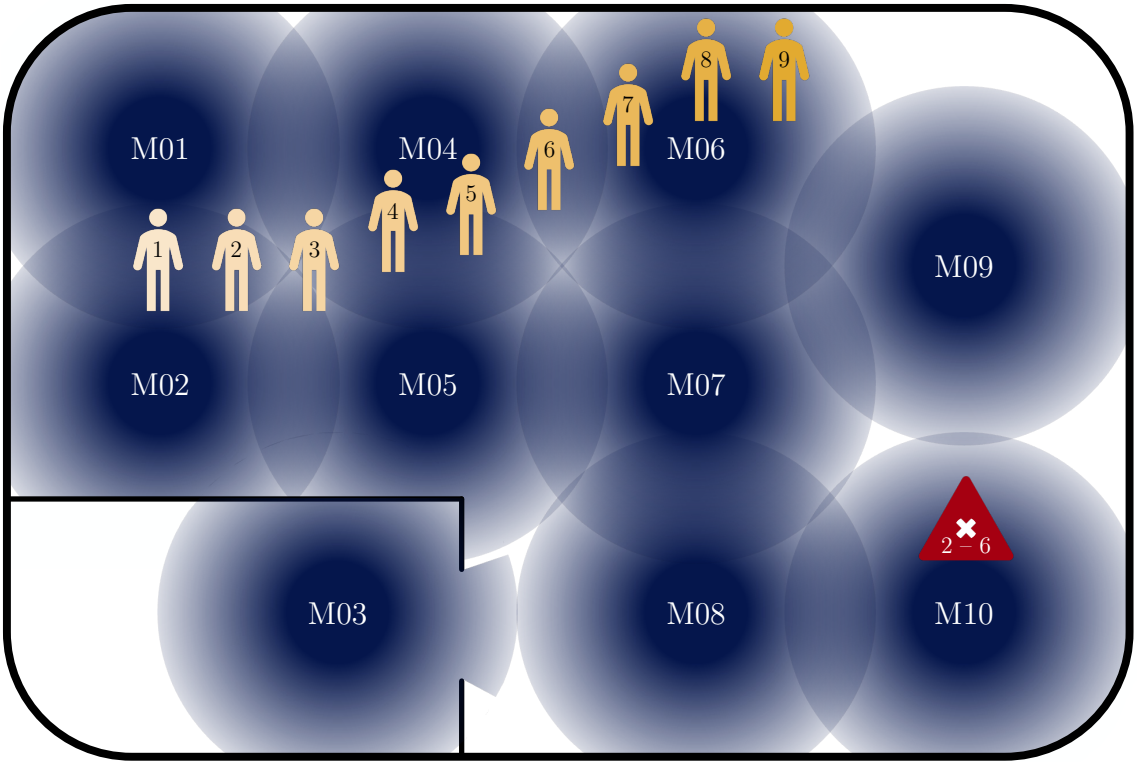


Abbildung 3.3: Umgebung mit Bewegungssensoren und einer Entität

dass jeder Fall möglichst genau eine Aktivität beinhaltet. Sind mehrere Aktivitäten oder nur Teile einer Aktivität in Fällen enthalten, kann dies negative Auswirkungen auf das resultierende Prozessmodell haben.

Zuordnung der Fälle: Eindeutige Identifizierung der Entitäten

In Abbildung 3.3 ist an einem vereinfachten Beispiel gezeigt, wie eine Person einen Bereich, ausgestattet mit Bewegungssensoren, durchläuft. Die Bewegungsmelder sind von $\mathtt{M}_0$ bis $\mathtt{M}_{10}$ durchnummeriert und in blau dargestellt. Die Erfassungsbereiche der einzelnen Sensoren sind durch die Kreisflächen um die Sensoren erkennbar. Mit steigendem Abstand zum Sensor sinkt die Wahrscheinlichkeit, dass eine Bewegung von dem entsprechenden Sensor erfasst wird, was durch den Farbgradienten von dunkelblau zu weiß veranschaulicht wird. Die Erfassungsbereiche der Sensoren sind nicht scharf voneinander getrennt. Eine Überlappung der Bereiche ist möglich. Sollten Überlappungen vorliegen, ist eine genauere Bestimmung der Position einer

Tabelle 3.3: Sensorereignislog zu Abbildung 3.3

Zeitstempel	Sensor ID	Wert	Einheit
24-01-01 00:00:01	$\mathtt{M}_{01}$	1	-
24-01-01 00:00:01	$\mathtt{M}_{02}$	1	-
24-01-01 00:00:01	$\mathtt{M}_{10}$	1	-
24-01-01 00:00:03	$\mathtt{M}_{04}$	1	-
24-01-01 00:00:03	$\mathtt{M}_{05}$	1	-
24-01-01 00:00:04	$\mathtt{M}_{01}$	0	-
24-01-01 00:00:04	$\mathtt{M}_{02}$	0	-
24-01-01 00:00:06	$\mathtt{M}_{05}$	0	-
24-01-01 00:00:06	$\mathtt{M}_{06}$	1	-
24-01-01 00:00:06	$\mathtt{M}_{10}$	0	-
24-01-01 00:00:07	$\mathtt{M}_{04}$	0	-

Entität möglich, da dann zwischen Überlappungsbereich (zwei oder mehrere Sensoren aktiv) und Kernerfassungsbereich des Sensors (nur ein Sensor ist aktiv) differenziert werden kann. Die Person ist beispielhaft in gelb dargestellt. Die Nummerierung von $1 - 9$ stellt die vergangene Zeit dar. Dies entspricht dem Bewegungsprofil der beobachteten Person.

Die aufgezeichneten Ereignisse der Sensoren zeigt Tabelle 3.3. Zum Zeitpunkt $t = 2$ ist der Sensor $\mathtt{M}_{10}$ bis $t = 2$ aktiviert. Hierbei handelt es sich um eine Fehlaufzeichnung (dargestellt durch das rote Dreieck in der Abbildung). Fehlauslösungen wie diese oder Auslösungen, die durch eine weitere Entität im beobachteten Bereich entstehen, führen zum Ausschluss des betroffenen Zeitfensters von der weiteren Auswertung.

Die Zuordnung von Sensorereignissen zu Fällen bei eindeutiger Entitätszuordnung ist in Algorithmus 3 dargestellt. Die Funktion gliedert sich folgendermaßen:

Zeile 2 – Initialisierung und Konfiguration: Der Code lädt zunächst verschiedene Einstellungen wie Pfade zu Datenquellen und Konfigurationen für das Ausgabeformat.

Zeile 3 – Sortierung SensorID: Der Sensor Ereignislog wird nach der SensorID und anschließend nach dem Zeitstempel sortiert. Diese doppelte Sortierung

Algorithmus 3 Ereignislog (Eindeutige Entität)

```
1: function ConvertRawDataToEventLog(sensor_data, distance_matrix)
2:     Initialisation
3:     Sort sensor_data by ['SensorID', 'Timestamp']
4:     for each event e ∈ sensor event E_L do
5:         Create column entry ['Sensor_Added']
6:         Remove invalid activations
7:         Calculate 'SensorActivationTime'
8:         Remove irrelevant sensor types
9:     end for
10:    Sort sensor_data by ['Timestamp']
11:    for each event e ∈ sensor event E_L do
12:        Calculate distance between sensor of this event and preceding event
13:        If no sensor is active add pseudo sensor M0 to event log
14:        Add Room/Area information
15:    end for
16:    LongGapsIndices: Detect long time gaps that separate time frames
17:    Identify multi-entity time frames based on sensor activation distances
18:    Finalising
19:    return sensor_data_cases
20: end function
```

stellt sicher, dass die Daten für jeden Sensor chronologisch geordnet sind. Durch diese methodische Anordnung der Daten wird es möglich, die Aktivierungsdauer der einzelnen Sensoren effizient zu berechnen. Diese Berechnung erfolgt durch numpy Spaltenoperationen, die über die sortierten Datensätze ausgeführt werden.

Zeilen 4 & 11 – For-Schleife: Die Schritte, die in der For-Schleife ab Zeile 4 des Pseudocodes beschrieben sind, werden im Python-Code, ohne den Einsatz herkömmlicher For-Schleifen durchgeführt. Stattdessen werden numpy-Spaltenoperationen verwendet, um die Laufzeit des Codes zu reduzieren. Mithilfe der numpy-Operationen ist es möglich, Operationen auf ganze Arrays gleichzeitig anzuwenden, was die Effizienz im Bezug auf die Laufzeit verglichen mit iterativen Methoden in Python drastisch erhöht. Durch den Einsatz von numpy-Spaltenoperationen wird die Effizienz, Skalierbarkeit und Lesbarkeit

des Python-Codes verbessert, wodurch eine schnellere und robustere Verarbeitung der Sensordaten möglich ist.

Zeile 5 – Sensorereignisse verarbeiten: In diesem Schritt wird eine Hilfsspalte ['Sensor_Added'] erstellt, in der der in dem jeweiligen Ereignis e hinzugefügte Sensor aufgeführt wird. Diese Spalte ist essenziell, um zwischen Zeilen mit Sensordeaktivierungen und solchen mit Sensoraktivierungen zu unterscheiden. Die Einführung dieser Hilfsspalte verbessert die Nachvollziehbarkeit der Ereignisprotokolle, da jedes Ereignis nun explizit angibt, welcher Sensor zu welchem Zeitpunkt aktiviert wurde. Dies erleichtert die Analyse und Verarbeitung der Sensordaten erheblich, da Zustandsänderungen der Sensoren klar differenziert werden können.

Zeile 6 – Bereinigung: In diesem Schritt wird der gesamte Ereignislog überprüft, um die Konsistenz der Sensoraktivierungen sicherzustellen. Es wird geprüft, ob für jeden Sensor zunächst eine Aktivierung stattgefunden hat und ob jede Sensoraktivierung mit einer Deaktivierung abgeschlossen wird. Einträge, die diesen Kriterien nicht erfüllen, werden als ungültig erkannt und entfernt. Unstimmige Sensoraktivierungen, wie beispielsweise wiederholte Aktivierungen ohne dazwischenliegende Deaktivierung, werden identifiziert und ebenfalls aus dem Log entfernt. Diese Bereinigung garantiert die Integrität des Sensoreregnislog. Durch diesen Validierungsschritt wird sichergestellt, dass die auszuwertenden Daten hinsichtlich Aktivierungs- und Deaktivierungsreihenfolge korrekt und verlässlich sind.

Zeile 7 – Aktivierungszeit: In diesem Teil wird die Aktivierungszeit der Sensoren berechnet. Dazu wird die Zeitspalte kopiert, der Index der Kopie um eine Position verschoben und anschließend die Differenz der Zeilenwerte berechnet. Diese Methode ermöglicht die Bestimmung der Dauer, während ein Sensor aktiviert ist. Die Berechnung erfolgt durch Bildung der Differenz zwischen Aktivierungs- und Deaktivierungszeitpunkt. Dies gewährleistet eine präzise zeitliche Analyse der Sensordaten.

Zeile 8 – Sensortypen: Hier werden nur die Sensortypen berücksichtigt, die für die Analyse von Interesse sind. Alle anderen für die weitere Auswertung irrelevanten Sensortypen werden aus dem Datensatz entfernt. Dies soll die Datenverarbeitung effizient ermöglichen.

Zeile 10 – Sortierung Zeitstempel: In Vorbereitung auf die kommenden Berechnungen wird die ursprüngliche Sortierung der Sensordaten nach Zeitstempel wiederhergestellt.

Zeile 12 – Distanzberechnung: Diese Zeile beschreibt die Berechnung der Distanz zwischen dem Ereignis zum Zeitpunkt e_t und dem unmittelbar vorangegangenen Ereignis e_{t-1}. Dies erfolgt mithilfe einer zuvor definierten Distanzmatrix, die die Abstände zwischen allen möglichen Sensoren enthält. Mit Hilfe der Distanzmatrix können die Abstände zwischen den Sensoren direkt verwendet werden, ohne erneut berechnet zu werden. Die jedem Ereignis zugeordnete Distanzwerte zum vorigen Ereignis e_{t-1} sind entscheidend für die Analyse von Bewegungspfaden und Aktivitätsmustern innerhalb des überwachten Bereichs.

Zeile 13 – Hilfssensor `M0`: Falls zu einem Zeitpunkt kein einziger Sensor aktiv ist, wird dies mit dem Hilfssensor `M0` im Sensorereignisprotokoll vermerkt. Der Hilfssensor `M0` dient dazu, Zeiträume eindeutig zu kennzeichnen, in denen keine Sensoraktivität vorliegt. Diese Auflistung der Zeiträume ohne jegliche Sensoraktivität ermöglicht später im Auswertungsverlauf ein rasches Auffinden ohne Inaktivitätszeiträume, ohne dass explizit weitere Berechnungen nötig wären. Die Einführung des Hilfssensors `M0` stellt außerdem sicher, dass das Ereignisprotokoll vollständig bleibt, selbst wenn keine Sensoraktivität vorliegt.

Zeile 14 – Raumzuordnung: Sofern ein Raumverzeichnis vorliegt, in dem alle Sensoren einem Raum oder Bereich zugeordnet sind, wird dieses benutzt, um jedem Ereignis die Information hinzuzufügen, in welchem Bereich das jeweilige Ereignis stattgefunden hat. In den Randbereichen von Räumen oder offenen Bereichen kann es durch Überlappungen der Erfassungsbereiche von Sensoren vorkommen, dass, obwohl die Entität den Bereich nicht gewechselt hat, ein Sensor aus dem anderen Bereich aktiviert wurde. Bei der Raumzuordnung der einzelnen Ereignisse zum Zeitpunkt t wird ebenfalls die Raumzugehörigkeit des aktivierten Sensors in den Ereignissen e_{t-1} und e_{t+1} überprüft. Sofern die aktivierten Sensoren in e_{t-1} und e_{t+1} dem gleichen Bereich zugeordnet sind, wird auch das Ereignis e_t dem gleichen Raum zugeordnet, unabhängig von der Bereichszugehörigkeit des Sensors aus e_t.

Zeile 16 – Inaktivität: In diesem Schritt werden lange Zeitlücken identifiziert und deren Indizes in einem Array mit der Bezeichnung *LongGapsIndices* gespeichert. Diese Zeitlücken treten auf, wenn für einen längeren Zeitraum keine

Sensoraktivität aufgezeichnet wurde. Die Definition dieser *langen* Zeiträume kann je nach Anwendungsdomäne variieren und ist entsprechend anpassbar. Die Identifizierung langer Inaktivitätsperioden ist von besonderer Bedeutung, da sie darauf hindeuten können, dass die beobachtete Entität den überwachten Bereich verlassen hat. Solche Zeitlücken liefern wertvolle Informationen über die Bewegungsmuster und Verhaltensweisen der Entität. Diese Informationen werden genutzt, um den gesamten Ereignislog in separate Abschnitte, sogenannte Zeitfenster (`timeframes`), zu unterteilen. Durch die Segmentierung des Ereignislogs in Zeitfenster wird eine detailliertere und strukturierte Analyse der Sensordaten ermöglicht. Jedes Zeitfenster repräsentiert einen zusammenhängenden Zeitraum, in dem kontinuierliche Sensoraktivität stattgefunden hat, getrennt durch Phasen der Inaktivität.

Zeile 17 – Mehrentitätsidentifikation: Für jedes der in Zeile 3 identifizierten Zeitfenster werden mögliche Multi-Entitäten Zeitfenster ermittelt. Dazu wird überprüft, ob Sensoraktivierungen auftreten, die aufgrund ihrer räumlichen Verteilung nicht durch die Bewegung einer einzigen Person erklärt werden können. Hierzu wird ein domänenspezifischer Schwellwert für die maximale Entfernung zwischen aktivierten Sensoren festgelegt. Überschreitet die Entfernung innerhalb eines Zeitfensters diesen Schwellwert, wird das gesamte Zeitfenster als Multi-Entitäten Zeitfenster markiert. Dieser Schritt ermöglicht die Identifikation von Zeitfenstern mit ungewöhnlich großen Abständen zwischen Sensoraktivierungen, was auf die Anwesenheit mehrerer Personen hinweist. Dies wiederum verbessert die Analyse der Bewegungsmuster und Interaktionen im überwachten Bereich.

Zeile 18 – Loggen und Speichern: Der Prozess wird zeitlich überwacht und die Verarbeitungszeit, sowie die verwendeten Parameter werden geloggt. Die Daten werden schließlich in einer CSV-Datei gespeichert.

Der Code nutzt Funktionen von Pandas, um die Daten zu verarbeiten, und stellt sicher, dass die Aktivitäten der Sensoren korrekt erfasst und umfassend dokumentiert werden. Dies ist besonders nützlich in Umgebungen, in denen Bewegung und Präsenz von Personen oder Objekten präzise verfolgt werden müssen.

Zuordnung der Fälle: Keine eindeutige Identifizierung der Entitäten

Sofern der Sensorereignislog Aktivitäten von mehreren Entitäten umfasst, ist im Folgenden beschrieben, wie aus einem solchen Sensorereignislog die Aktivitäten individueller Entitäten zugeordnet werden können. In intelligenten IoT-basierten Umgebungen ist es nicht ungewöhnlich, dass sich dort mehrere Entitäten gleichzeitig bewegen bzw. aufhalten.

Ereignisse sind in einem Sensorereignislog nicht unbedingt mit einer eindeutigen Kennung versehen, die angibt, von welcher Entität sie erzeugt wurden. Oftmals enthalten die Daten sich überschneidende und gleichzeitig stattfindende Aktivitäten von mehreren Personen/Objekten. Sensoren liefern in diesem Fall nur Informationen darüber, ob eine Person/Objekt im Beobachtungsbereich anwesend ist oder nicht, ohne eine Unterscheidung zwischen Entitäten zu ermöglichen.

Der nachfolgende Abschnitt beschreibt eine Heuristik, mit deren Hilfe es möglich ist, Entitäten voneinander zu unterscheiden. Sofern die relativen Positionen der Sensoren untereinander bekannt sind, ist eine Unterscheidung mehrerer Entitäten möglich. Das Vorgehen dazu ist in Algorithmus 4 beschrieben.

Sobald zum ersten Mal die Anwesenheit einer Entität durch einen Sensor registriert wird, beginnt die Trace der ersten Entität. Für jede nachfolgende Sensoraktivierung wird entschieden, ob die Aktivierung von der bereits betrachteten Entität ausgelöst wurde, oder ob eine neue Trace für eine neue Entität eröffnet wird.

Um zu entscheiden, ob eine Sensoraktivierung zu einer bereits existierenden Trace passt, wird die Distanz der Sensoren untereinander genutzt: Je näher ein Sensor den bisher aktivierten Sensoren einer Trace ist, desto wahrscheinlicher ist es, dass die Aktivierung Teil dieser Trace ist. Da es vorkommen kann, dass mehrere Sensoren von einer Person in einer Trace gleichzeitig aktiviert sind, muss bei der Zuordnung eines neuen Sensors die Entfernung zu allen aktuell aktiven Sensoren der Trace ermittelt werden. Der daraus gebildete Mittelwert gibt eine gute Einschätzung, welcher Trace und damit Entität der Sensor am nächsten ist. Falls keine Entität nahe genug ist, nimmt der Algorithmus an, dass eine Entität das beobachtete Gebiet neu betreten hat, und es wird eine neue Trace für eine neue Entität erstellt. Sowohl der Schwellenwert für die Annäherung als auch die maximale Anzahl der Entitäten sind Parameter, die je nach Szenario angepasst werden können.

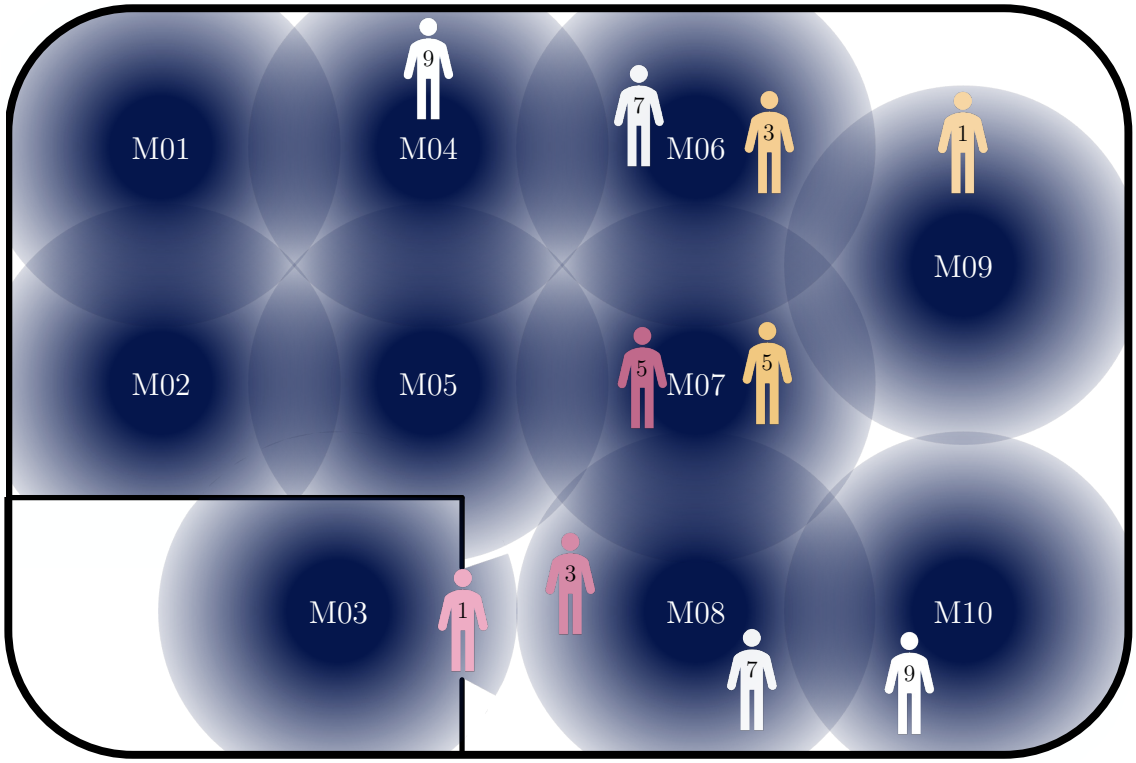


Abbildung 3.4: Umgebung mit Bewegungssensoren und zwei Entitäten.

Dieser simple Ansatz der Implementierung funktioniert hinreichend gut, solange Entitäten einen gewissen Abstand zueinander einhalten. Sobald sich die Wege der Entitäten an einer Stelle kreuzen (Abbildung 3.4 und korrespondierendes Sensorereignislog Tabelle 3.4), kann diese Methode die Sensoraktivierungen nach der Begegnung der Entitäten nicht mehr korrekt zuordnen.

So sehen in dem Beispiel aus dem Ereignislog 3.4 und Abbildung 3.4 die Traces beider Entitäten zum Zeitpunkt $t = 5$ folgendermaßen aus:

$$\tau_1 = (\texttt{M03}, \texttt{M08}, \texttt{M07})$$
$$\tau_2 = (\texttt{M09}, \texttt{M06}, \texttt{M07})$$

Die nächsten Sensoren, die im Sensorereignislog vorkommen, sind zum Zeitpunkt $t = 7$ `M06` und `M08`. Die neu aktivierten Sensoren haben jeweils den gleichen Abstand

Tabelle 3.4: Sensorereignislog zu Abbildung 3.4

Zeitstempel	Sensor ID	Wert	Einheit
24-01-01 00:00:01	$\mathtt{M}_{03}$	1	-
24-01-01 00:00:01	$\mathtt{M}_{09}$	1	-
24-01-01 00:00:03	$\mathtt{M}_{09}$	0	-
24-01-01 00:00:03	$\mathtt{M}_{06}$	1	-
24-01-01 00:00:03	$\mathtt{M}_{03}$	0	-
24-01-01 00:00:03	$\mathtt{M}_{08}$	1	-
24-01-01 00:00:05	$\mathtt{M}_{08}$	0	-
24-01-01 00:00:05	$\mathtt{M}_{06}$	0	-
24-01-01 00:00:05	$\mathtt{M}_{07}$	1	-
24-01-01 00:00:05	$\mathtt{M}_{07}$	1	-
24-01-01 00:00:07	$\mathtt{M}_{06}$	1	-
24-01-01 00:00:07	$\mathtt{M}_{08}$	1	-
24-01-01 00:00:07	$\mathtt{M}_{07}$	0	-
24-01-01 00:00:07	$\mathtt{M}_{07}$	0	-
24-01-01 00:00:09	$\mathtt{M}_{04}$	1	-
24-01-01 00:00:09	$\mathtt{M}_{06}$	0	-
24-01-01 00:00:09	$\mathtt{M}_{10}$	1	-
24-01-01 00:00:09	$\mathtt{M}_{08}$	0	-

($d(x, y)$ ist die Abstandsfunktion für die Sensoren x, y) zum zuletzt aktivierten Sensor beider Traces:

$$d(\mathtt{M07}, \mathtt{M06}) = 1 = d(\mathtt{M07}, \mathtt{M08})$$
$$d(\mathtt{M07}, \mathtt{M06}) = 1 = d(\mathtt{M07}, \mathtt{M08})$$

Diese Einschränkung kann umgangen werden, indem auch vergangene Aktivierungen miteinbezogen werden. Wenn man davon ausgeht, dass die Entitäten ihre Bewegungsrichtung beibehalten, kann berechnet werden, wohin die Entitäten gehen werden.

Tabelle 3.5: Beispiel zur Verdeutlichung des gewichteten Mittelwerts

Trace ID	**Sensoren**	$\frac{1}{1}\sum_{i=1}^{1} d(x, \tau[n-i])$		$\frac{1}{2}\sum_{i=1}^{2} \frac{i}{2} \times d(x, \tau[n-i])$	
		$x = \texttt{M06}$	$x = \texttt{M08}$	$x = \texttt{M06}$	$x = \texttt{M08}$
Trace rot:	[M08, M07]	1.0	1.0	1.5	0.5
Trace gelb:	[M06, M07]	1.0	1.0	0.5	1.5

Dies wird umgesetzt, indem man auch die vorherige Position der Entitäten berücksichtigt. Die Zerfallskonstante gewichtet weiter zurückliegende Ereignisse weniger stark als neu auftretende Ereignisse. Das Konzept des gewichteten Mittelwerts ist in Algorithmus 4 umgesetzt. Für jede Entität beziehungsweise den jeweiligen Trace wird der gewichtete Mittelwert für alle zurzeit aktiven Sensoren mithilfe des Gewichtungsfaktors $\lambda = \frac{i}{2}$ gebildet. Es wird überprüft, bei welchem der Sensoren der gewichtete Mittelwert höher ist und dann den entsprechenden Traces zugeordnet. Ist dies der Fall, wird der neue Sensor allen offenen Traces zugeordnet. In Abbildung 3.5 sind die Figuren der entsprechenden Zugehörigkeit der Traces eingefärbt, wie sie laut Tabelle 3.5 zugeordnet werden soll. Gibt es keine Trace, die dem neuen Sensor zugeordnet werden kann, wird davon ausgegangen, dass der neue Sensor von einer neuen Entität aktiviert wurde. Sofern die zuvor festgelegte Maximalanzahl der Personen im Haus nicht überschritten wird, wird nun eine neue Trace für eine neue Entität erstellt. Ist die Schwelle der maximal zulässigen Entitäten in der beobachteten Umgebung überschritten, wird der neue Sensor der Trace zugeordnet, die unter Berücksichtigung des gewichteten Mittelwerts am nächsten ist. In Abbildung 3.5 sind die Sensoren der jeweiligen Entität zugeordnet und eingefärbt.

Ein Nachteil dieser Methode ist die Annahme einer kontinuierlichen Bewegungsrichtung. Sobald die Entitäten am Ort der Begegnung nicht nur aneinander vorbeigehen, sondern länger miteinander interagieren und eventuell danach die ursprüngliche Bewegungsrichtung nicht beibehalten, liefert der Ansatz möglicherweise nicht die gewünschten Ergebnisse. Für eine bessere Identifizierung von Entitäten könnten Hidden Markov Modelle hinzugezogen werden, welche eine bessere Unterscheidung der Bewegungsrichtung ermöglichen (Guo and Miao 2010).

Algorithmus 4 Entitäten identifizieren

```
1: function DistinguishResidents(raw_data_sensor)
2:     if open_individual_traces = ∅ then
3:         open_individual_traces[unique_trace_id] ← raw_data_sensor
4:         unique_trace_id ← unique_trace_id + 1
5:     else
6:         for each trace_i in open_individual_traces do
7:             λ ← 1                                   ▷ weighted decay factor
8:             sum ← 0
9:             for each sensor in last sensor data of trace_i do
10:                sum ← sum + λ × distance(sensor, raw_data_sensor)
11:                λ ← λ + 1
12:            end for
13:            average_distance_i ← sum / (Σ_{n=1}^{λ−1} n)
14:        end for
15:        min_distance, min_trace ← min_i(average_distance_i)
16:        if min_distance ≤ distance_threshold then
17:            append raw_data_sensor to min_trace
18:        else
19:            if |open_individual_traces| < max_people then
20:                open_individual_traces[unique_trace_id] ← raw_data_sensor
21:                unique_trace_id ← unique_trace_id + 1
22:            else
23:                append raw_data_sensor to trace with smallest average_distance_i
24:            end if
25:        end if
26:    end if
27: end function
```

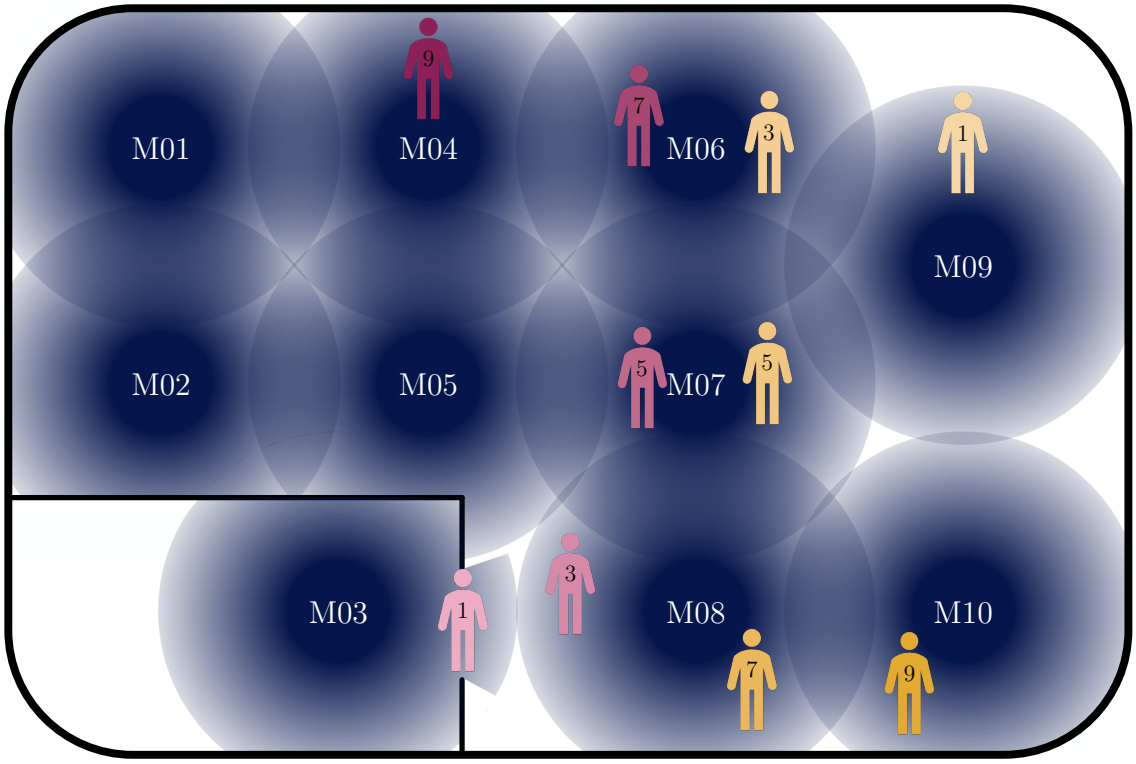


Abbildung 3.5: Umgebung mit Bewegungssensoren und zwei Entitäten.

Partitionierung der Fälle

Im vorigen Schritt, der Zuordnung der Fälle, wurde ein Fall kontinuierlicher Sensoraktivierungen erstellt. Dieser ist zwar unterbrochen, sobald die Entität das beobachtete Gebiet für eine gewisse, zuvor festgelegte Zeit verlässt, jedoch beinhaltet dieser Fall unter Umständen mehrere Aktivitäten. Denn während sich eine Entität in einem beobachteten Gebiet befindet, ist nicht auszuschließen, dass die Entität mehr als eine Aktivität ausführt. Da Aktivitäten aber anhand ihrer Signatur von Sensoraktivierungen identifiziert und geclustert werden, ist es notwendig, aus der kontinuierlichen Aufzeichnung einzelne Fälle mit einzelnen Aktivitäten zu identifizieren.

Konkret wird in dem Ansatz eine lange Abfolge an Sensoraktivierungen in Fälle mit einer vordefinierten festen Länge zerlegt. Je nach Anwendung kann die optimale Länge jedoch unterschiedlich sein. Die Hyperparametersuche berücksichtigt verschiedene Längen von *Fällen* und vergleicht die Ergebnisse verschiedener Längen von Fällen. Dies stellt sicher, dass der Ansatz sich auf verschiedene Szenarien anpassen lässt. Die größte Herausforderung dieses Schrittes besteht darin, zu kurze Fälle

zu vermeiden. Ein zu kurzer Fall enthält im Extremfall nur noch eine Sensoraktivierung, die nur noch beschränkt Rückschlüsse auf die zugrunde liegende Aktivität erlaubt. Gleichzeitig dürfen die Fälle nicht zu lang sein, da ein zu langer Fall, wie oben beschrieben, mehrere Aktivitäten enthalten kann.

Hierfür werden drei mögliche Arten der Aufteilung (Partitionierungen) verwendet:

Aktivierungsanzahl Die Sensorereignisdaten werden in Fälle mit einer festen Anzahl von aktivierten Sensoren aufgeteilt. Angenommen, es liegt ein Sensorereignislog mit n Einträgen von Sensorereignisdaten vor. Wenn die Anzahl der erlaubten Aktivierungen pro Fall i beträgt, wird das Sensorereignislog in $\frac{n}{i}$ Fälle aufgeteilt.

Aktivierungszeit Dieser Modus teilt das Sensorereignislog nach Aktivierungszeiten auf. Der Modus der Aktivierungszeit nimmt an, dass alle Aktivitäten die identische Zeit benötigen. Der Benutzer legt ein Zeitlimit t_{max} fest. Neue Sensorereignisse werden einem Fall hinzugefügt, solange der Zeitunterschied zwischen dem ersten Ereignis im Fall t_1 und dem hinzugefügten Ereignis mit dem Zeitstempel t_i kleiner als eine Schwelle ist ($t_i - t_1 \leq t_{max}$). Sobald die Schwelle überschritten wird, beginnt ein neuer Fall.

Örtliche Begrenzung Bei der örtlichen Begrenzung (z.B. Wohnzimmer, Küche oder Schlafzimmer) sind die Aktivitäten auf einen Raum oder Bereich beschränkt. Verlässt eine Entität diesen vordefinierten Bereich, endet die aktuelle Aktivität, und eine neue beginnt. Diese Annahme schließt Aktivitäten in mehreren Räumen aus.

Die vorgestellten Arten der Aufteilung haben aufgrund ihrer Einfachheit jedoch auch Schwächen: Betrachtet man die beiden Aktivitäten *Schlafen* und *Essenszubereitung*, findet *Schlafen* über einen längeren Zeitraum statt, die Anzahl der beteiligten Sensoren ist jedoch deutlich geringer als bei der *Essenszubereitung*.

Algorithmus 5 beschreibt, wie die Partitionierung des Sensorereignislogs im Code umgesetzt wurde. Dem Algorithmus 5 werden vier relevante Variablen übergeben:

sensor_data_cases: Dieses Array enthält den vorverarbeiteten Ereignislog aus Algorithmus 3 bzw. Algorithmus 4. Der Ereignislog enthält nun zusätzlich die Zuordnung von den Ereignissen zu Entitäten. Außerdem ist der Ereignislog von Sensoraktivierungen um Rauschen bereinigt.

partition_method: Parameter, der bestimmt, ob der Fall nach Aktivierungszeit oder Häufigkeit der Aktivierung partitioniert werden.

multi_room_activity: Dieser Parameter entscheidet, ob Aktivitäten über fest definierte, räumliche Bereiche hinausgehen dürfen, oder ob Aktivitäten auf einen einzelnen Bereich beschränkt werden.

number_time_per_case: Parameter der festlegt, wie viele Aktivierungen maximal in einem Fall enthalten sind, bzw. wie lange die Dauer eines Falles maximal sein darf.

Algorithmus 5 Partitionierung der Fälle

```
1: function PartitionLogIntoCases(sensor_data_cases, partition_method,
                                  multi_room_activity,
                                  number_time_per_case)
2:     Initialize logger
3:     for each event_i in event_log do
4:         event_i[case] ← i
5:         i ← i + 1
6:     end for
7:     if partition_method = 'FixedSensorActivations' then
8:         Check if method previously executed with same parameters
9:         if not previously executed then
10:            if multi_room_activity then
11:                for each event_i in event_log do
12:                    event_i[case] ← no_of_activations × event_i[TimeFrame] − 1
13:                    event_i[case] ← ⌊(event_i[case] − 1) ÷ no_of_activations⌋ + 1
14:                end for
15:            else
16:                room_change_count ← 0
17:                for each event_i in event_log do
18:                    if event_i[room] ≠ event_{i−1}[room] then
19:                        room_change_count ← room_change_count + 1
20:                        event_i[room_change] ← room_change_count
21:                        Begin new case at room change
22:                    end if
23:                end for
24:            end if
25:        end if
```

Die Funktion gliedert sich folgendermaßen:

Zeile 2 – Initialisierung und Konfiguration: Der Code lädt zunächst verschiedene Einstellungen, wie Pfade zu Datenquellen und Konfigurationen für das Ausgabeformat.

Zeile 4 – Fälle initialisieren: Jedes Ereignis im Ereignislog, das eine Sensoraktivierung enthält, wird zunächst einem eigenen Fall zugeordnet. Durch die initiale 1:1-Zuordnung zwischen Fallnummer und Sensoraktivierung können später durch arithmetische Operationen auf die Fallnummern die tatsächlichen Fallnummern effizienter berechnet werden.

Zeilen 7 & 26 – Pratitionsmethode: Je nach Wahl der Partitionsmethode (Aktivierungsdauer oder Anzahl der Aktivierungen) wird der entsprechende Codeblock ausgeführt.

Zeile 8 & 27 – Redundanzprüfung: Durch die Hyperparameteroptimierung ist es möglich, dass in einer vorherigen Iteration die Einteilung des Ereignislogs in Fälle bereits mit der exakt gleichen Parameterkombination durchgeführt wurde. Die Zeile des Algorithmus überprüft dies. Ist dies der Fall, wird auf das Ergebnis der vorigen Iteration zurückgegriffen. Somit wird verhindert, dass gleiche Berechnungen mehrfach durchgeführt werden müssen.

Zeilen 10 & 15 – Raumbeschränkung: An diesen Stellen wird überprüft, ob eine Betrachtung von Aktivitäten über festgelegte räumliche Begrenzungen gestattet ist. Entsprechend der Parameterwahl wird der jeweilige Codeblock ausgeführt.

Zeile 12 – Berechnung Fall: Der Eintrag `timeframes`, welcher in Algorithmus 3 Zeile 16 bestimmt wurde, enthält eine fortlaufende ganze Zahl, die sich um eins erhöht, wenn zwischen zwei Ereignissen ein längerer (zuvor festgelegter) Zeitraum liegt. Sollte ein längerer Zeitraum zwischen zwei Ereignissen auftreten, kann davon ausgegangen werden, dass es sich um zwei verschiedene Fälle handelt.

Dies wird erreicht, indem für jede Zeile im DataFrame, bei der die Spalte 'Active' den Wert eins hat, der Wert in der Spalte 'Case' um das Produkt der Anzahl der Aktivierungen ($number\_of\_activations$) und des Zeitintervalls ($event_i['TimeFrame'] - 1$) erhöht. Das bedeutet, dass jedes Mal, wenn ein

großer Zeitraum zwischen den Aktivitäten festgelegt wird, die Fallzahl entsprechend angepasst wird.

Zeile 13 – Berechnung Fall: Hier wird jede Fall-Nummer durch die zuvor festgelegte maximale Länge der Fälle geteilt. Dabei wird ein Gleitkommawert zurückgegeben, von dem nur der ganzzahlige Wert beibehalten und um eins erhöht wird.

Es wird zunächst von der Fallzahl eins subtrahiert und das Ergebnis durch die Anzahl der Aktivierungen ($number\_of\_activations$) geteilt. Das Resultat ist ein Gleitkommawert, der mithilfe der Abrundungsfunktion in einen ganzzahligen Wert umgewandelt wird. Anschließend wird dieser Wert um eins erhöht. Diese Berechnung wird ebenfalls nur auf diejenigen Zeilen (Ereignisse) angewendet, bei denen der Sensor aktiv ist.

Zeilen 18 bis 20 – Raumwechsel überprüfen: Um die Berechnung der Fallnummer für nicht zulässige Aktivierungen bei Raumwechsel zu beschleunigen, wird eine Hilfsvariable `room_change` eingeführt. Diese wird um eins erhöht, sobald sich der Raum zwischen nachfolgenden Ereignissen verändert.

Zeile 21 – Berechnung Fall: Sobald ein Raumwechsel zwischen zwei aufeinanderfolgenden Ereignissen vorkommt, wird der aktuelle Fall beendet und mit dem Raumwechsel beginnt ein neuer Fall.

Zeile 31 – Zeitrest berechnen (Zeit & Multi Room): Dieser Schritt berechnet für jedes Zeitfenster (`timeframes`), welcher Zeitrest verbleibt, wenn das Zeitfenster durch die zuvor festgelegte maximale Zeitdauer für einen Fall geteilt wird.

$$remainder(TimeFrame_i) = duration(TimeFrame_i) \bmod duration\_per\_case$$

Zeile 32 – Zeitrest aufaddieren (Zeit & Multi Room): Der in Zeile 31 berechnete Zeitrest wird zu den letzten Ereignissen jedes Zeitfensters aufaddiert.

Algorithmus 5 Partitionierung der Fälle (Fortsetzung)

```
    else if partition_method = 'FixedActivationTime' then
        Check if method previously executed with same parameters
        if not previously executed then
            if multi_room_activity then
                for each event_i in event_log do
                    Calculate cumulative time remainder per TimeFrame
                    Add remainder to last event in TimeFrame
                    j ← index of the last event in the previous case
                    event_i[case] ← ⌊((Σ_{k=j}^{i} event_k[duration] − ϵ) ÷
                                                    CaseDuration) + 1⌋
                end for
            else
                for each event_i in event_log do
                    Calculate cumulative time remainder per room change
                    Add remainder to last event in room
                    j ← index of the last event in the previous case
                    event_i[case] ← ⌊((Σ_{k=j}^{i} event_k[duration] − ϵ) ÷
                                                    CaseDuration) + 1⌋
                end for
            end if
        end if
    end if
    Normalize case IDs to ensure consecutive numbering
    if not previously executed then
        Export processed events to file
    else
        Import existing processed events
    end if
    return event_log_cases
end function
```

Zeile 34 – Fallnummer berechnen (Zeit & Multi Room): Die Fallnummer kann nun durch eine einfache Division mit anschließendem Abrunden nach folgender Formel berechnet werden:

$$event_i[case] = \lfloor (\frac{\Sigma_{k=j}^{i} event_i[duration] - \epsilon}{CaseDuration} + 1 \rfloor$$

j sei hierbei der Index des letzten Ereignisses des vorhergehenden Falles, mit $j < i$. Der Parameter $CaseDuration$ ist die zuvor festgelegte fixe Größe des Zeitfensters.

Zeile 41 – Fallnummer berechnen (Zeit & Single Room): Die Berechnung der Fallnummer kann analog zu den Berechnungen von Zeile 34 erfolgen.

Zeile 46 – Nummerierung der Fälle: Die Berechnung der Fallnummer durch Division in den vorigen Schritten stellt zwar sicher, dass jeder Fall (bestehend aus mehreren Ereignissen) eine eindeutige Fallnummer zugewiesen bekommt. Diese ist jedoch nicht zwingend eine fortlaufende ID. In diesem Schritt wird nun allen Fällen eine fortlaufende Fall-ID zugewiesen.

Fall Einteilung Die Logik der Einteilung der Fälle ist auf zwei Arten möglich. Die Länge der Fälle kann entweder durch die Anzahl der Sensoraktivierungen bestimmt werden oder durch die Gesamtaktivierungsdauer der Sensoren.

Festgelegte Sensoraktivierungen (FixedSensorActivations): Diese Art teilt die Sensorprotokolle basierend auf einer festgelegten Anzahl von Sensoraktivierungen pro Fall. Sobald eine zuvor festgelegte Anzahl an Aktivierungen erreicht wird, wird der Fall als vollständig betrachtet.

Festgelegte Aktivierungszeit (FixedActivationTime): Sobald die kumulierte Aktivierungsdauer aller Sensoren in einem Fall einen vorher festgelegten Schwellenwert erreicht hat, wird der Fall als vollständig betrachtet und abgeschlossen.

Raumwechsel: Ja nach Anwendungsdomäne kann entschieden werden, Aktivitäten über mehrere Räume zu erlauben oder Aktivitäten auf einen begrenzten Bereich zu beschränken. Sind Aktivitäten nicht über mehrere Räume erlaubt, wird ein Fall sofort als abgeschlossen markiert, sobald die Entität den vorher festgelegten Bereich verlässt, unabhängig davon, wie viele Sensoraktivierungen

der Fall bisher beinhaltet und unabhängig von der kumulierten Aktivierungszeit des Falls.

Ein Beispiel für die Aufteilung wird in Tabelle 3.6 gezeigt. Für das Beispiel in der Tabelle sind die Partitionierungsparameter wie folgt gewählt:

Festgelegte Sensoraktivierungen (FixedSensorActivations)	5 Aktivierungen
Festgelegte Aktivierungszeit (FixedActivationTime)	5 Minuten

Festgelegte Sensoraktivierungen In der Spalte *Aktivierungsanzahl* ist die Begrenzung von 5 Aktivierungen relevant. Sobald die Grenze von fünf Aktivierungen erreicht wird, gilt der Fall als abgeschlossen und ein neuer Fall wird begonnen. Deaktivierungen spielen bei dieser Zählweise keine Rolle. Ausschließlich die Ereignisse von aktivierten Sensoren werden gezählt. Es ergibt sich so die folgende Aufteilung der Sensoraktivierung auf die einzelnen Fälle:

1_{Akt}	M22, M23, M01, M07, M06
2_{Akt}	M05, M06, M04, M08, M01
3_{Akt}	M10, M11, M12, M13, M14

Festgelegte Aktivierungszeit In der Spalte *Aktivierungszeit* ist die Begrenzung von 5 Minuten relevant. Sobald diese Obergrenze überschritten wird, gilt die Aufteilung als abgeschlossen. Für jede neue Sensoraktivität wird die verstrichene Zeit errechnet, es spielt hierbei keine Rolle, ob es sich um eine Aktivierung oder Deaktivierung eines Sensors handelt. So wird im Beispiel um 13:10:15 der Sensor M04 deaktiviert und damit die fünf Minuten-Schwelle überschritten. Mit diesem Ereignis wird dann der Fall 2_{Zeit} abgeschlossen. Die daraus resultierende Fallaufteilung ist folgendermaßen:

1_{Zeit}	06:42 Minuten	M22
2_{Zeit}	05:03 Minuten	M23, M01, M07, M06, M05, M06, M04, M08
3_{Zeit}	05:01 Minuten	M01, M10, M11, M12, M13, M14

In Tabelle 3.6 sind die Gruppierungen der Ereignisse in Fälle durch die gestrichelten Linien in der jeweiligen Spaltengruppe eingezeichnet. In der Spaltengruppe *Aktivierungsanzahl* umfassen die gestrichelten Kästen nicht die Deaktivierungen, da diese für die Zählweise und damit die Fallbestimmung irrelevant sind.

3.3 Clustering

Ausgangslage für das in diesem Schritt beschriebene Clustering sind die partitionierten Fälle aus dem erweiterten Ereignislog, wie in Kapitel 3.2 beschrieben. Zum jetzigen Zeitpunkt sollte im Ereignislog jeder Fall nur noch genau eine Aktivität beschreiben. Für das Clustering sind zwei Schritte nötig:

Fälle vektorisieren: Die Anwendbarkeit der in dieser Arbeit genutzten Clustering-Algorithmen setzt voraus, dass zwischen den einzelnen Fällen ein aussagekräftiges Distanzmaß gebildet werden kann. Dies ist am besten durch das Vektorisieren der Fälle möglich.

Clustering-Algorithmus anwenden: Das Ziel des Clusterings besteht darin, ähnliche Fälle demselben Cluster zuzuordnen. Jeder Cluster repräsentiert eine Aktivität (z.B. *Sitzen*, *Essen*). Der vorgestellte Ansatz unterstützt drei Clustering-Algorithmen, deren Funktionsweise in Kapitel 2.7 vorgestellt wurde: k-Means, k-Medoids und Self-Organising Map (SOM).

Die Herausforderungen hierbei sind, die Aufteilung der Traces nach Clustern zu bestimmen und eine geeignete Anzahl von Clustern zu ermitteln. Darüber hinaus muss eine geeignete Aktivitätskennzeichnung gefunden werden. In den folgenden Unterkapiteln wird die Vorgehensweise für beide Schritte erläutert:

Vektorisierung der Fälle

Fälle mit Sensorereignissen zu clustern, hat die folgenden Herausforderungen, die es zu überwinden gilt:

Unterschiedliche Länge der Fälle: Die betrachteten Fälle haben nicht zwingend die gleiche Anzahl an Ereignissen. Unterschiedlich lange Vektoren erschweren das Anwenden von Clustering-Algorithmen. Überwunden werden können die Längenunterschiede durch eine Transformation bzw. Codierung

Tabelle 3.6: Beispiel der Zeit- und Anzahlaufteilung.

			Aktivierungsanzahl		Aktivierungszeit	
Zeit	**SensorID**	**Status**	**Zähler**	**Partition**	**Dauer**	**Partition**
12:59:57	M22	ON	1	1_{Akt}	0:00	1_{Zeit}
13:06:21	M22	OFF		1_{Akt}	6:24	1_{Zeit}
13:06:22	M23	ON	2	1_{Akt}	0:00	2_{Zeit}
13:06:23	M01	ON	3	1_{Akt}	0:01	2_{Zeit}
13:06:24	M23	OFF			0:02	2_{Zeit}
13:06:24	M01	OFF			0:02	2_{Zeit}
13:06:24	M07	ON	4	1_{Akt}	0:02	2_{Zeit}
13:06:30	M07	OFF			0:08	2_{Zeit}
13:06:30	M06	ON	5	1_{Akt}	0:08	2_{Zeit}
13:06:33	M06	OFF			0:11	2_{Zeit}
13:06:33	M05	ON	1	2_{Akt}	0:11	2_{Zeit}
13:06:35	M06	ON	2	2_{Akt}	0:13	2_{Zeit}
13:06:45	M06	OFF			0:23	2_{Zeit}
13:10:15	M05	OFF			04:53	2_{Zeit}
13:10:15	M04	ON	3	2_{Akt}	04:53	2_{Zeit}
13:10:20	M08	ON	4	2_{Akt}	04:58	2_{Zeit}
13:10:25	M04	OFF			05:03	2_{Zeit}
13:14:00	M01	ON	5	2_{Akt}	00:00	3_{Zeit}
13:14:25	M08	OFF			00:25	3_{Zeit}
13:15:29	M10	ON	1	3_{Akt}	01:29	3_{Zeit}
13:15:30	M01	OFF			01:30	3_{Zeit}
13:15:50	M11	ON	2	3_{Akt}	01:50	3_{Zeit}
13:16:55	M12	ON	3	3_{Akt}	02:55	3_{Zeit}
13:16:56	M10	OFF			02:56	3_{Zeit}
13:17:12	M13	ON	4	3_{Akt}	03:12	3_{Zeit}
13:17:20	M11	OFF			03:20	3_{Zeit}
13:17:44	M12	OFF			03:44	3_{Zeit}
13:18:18	M14	ON	5	3_{Akt}	04:18	3_{Zeit}
13:18:47	M13	OFF			04:47	3_{Zeit}
13:19:01	M14	OFF			05:01	3_{Zeit}

der Ereignisse. Alternativ können die Vektoren durch Auffüllen (*padding*) in der Länge so angeglichen werden, dass alle Vektoren die gleiche Länge wie der längste vorkommende Vektor zugewiesen bekommen.

Distanz zwischen Fällen: Die Herausforderung bei der Entdeckung von Fällen, die sich ähneln und die in eine Gruppe geclustert werden können, besteht darin, ein Kriterium zu finden, um die Ähnlichkeit zwischen den jeweiligen Fällen zu definieren. Clusteringverfahren bilden Cluster üblicherweise basierend auf einer Distanzfunktion zwischen einzelnen Vektoren bzw. zwischen Vektoren und Clusterschwerpunkten. In Ereignislogs mit Sensordaten sind die Ereignisse oftmals mit Sensornamen bezeichnet. Diese folgen möglicherweise einer Ordnung oder einem systematischen Bezeichnungssystem. Die Definition einer Distanzbeziehung zwischen den einzelnen Sensoren, die ausschließlich auf der Sensorbezeichnung basieren, ist selten realisierbar.

Um dieser Herausforderung zu begegnen, gibt es entweder die Möglichkeit, eine benutzerdefinierte Distanzfunktion in der Berechnung der Vektordistanz zu integrieren, oder die Vektoren nach Sensor-IDs zu transformieren. Diese Transformation soll ermöglichen, dass der euklidische Abstand oder andere Distanzmetriken sinnvolle Ergebnisse liefern.

Um die oben genannten Herausforderungen zu überwinden, werden die Fälle zu Vektoren mit gleicher Länge vereinheitlicht. Hierzu wird jedem Sensor eine eindeutige ganzzahlige Zahl beginnend bei $k = 1$ zugewiesen. Die Länge der Vektoren ist bestimmt durch die Anzahl der sich im beobachteten Bereich befindlichen Sensoren: Zusätzlich zu den Sensoren, die tatsächlich im Bereich vorkommen, gibt es noch den hypothetischen Hilfssensor $\mathtt{M}_0$ (eingeführt in Kapitel 3.1), der die Zeit bzw. Aktivierungen zählt, in denen kein (Bewegungs)sensor aktiviert ist. Hieraus ergibt sich dann bei k Sensoren im beobachteten Bereich eine Länge der Fall-Vektoren je nach Vektorisierungsmethode von $k + 1$ bzw. $2 \times (k + 1)$.

Das Framework unterstützt drei verschiedene Arten, die Fall-Vektoren zu transformieren:

- Anzahl
- Zeit
- Anzahl & Zeit

Diese Transformationen werden im Folgenden erklärt und zur Veranschaulichung an folgendem Beispiel mit den Sensoraktivierungen der Sensoren S_1 bis S_7, eines Falles beispielhaft gezeigt:

Zeitstempel	**Sensor**	**Aktivierungsdauer**
t_1	$\mathtt{S_1}$	d_1
t_2	$\mathtt{S_7}$	d_2
t_3	$\mathtt{S_6}$	d_3
t_4	$\mathtt{S_5}$	d_4
t_5	$\mathtt{S_6}$	d_5

Die in der oben stehenden Tabelle dargestellten Sensoraktivierungen können folgendermaßen zu einem Vektor transformiert werden:

Anzahl: Bei dem Transformationsmodus *Anzahl* wird gezählt, wie oft jeder Sensor in einem Fall aktiviert wurde. Wird ein Sensor nicht aktiviert (beispielsweise Sensor $\mathtt{S_2}$), resultiert dies in einer 0 an der entsprechenden Stelle des Vektors. In diesem Beispiel bei Index 2:

$$\begin{aligned}\tau_1^Q &= (\#\mathtt{S_1}, \#\mathtt{S_2}, \#\mathtt{S_3}, \#\mathtt{S_4}, \#\mathtt{S_5}, \#\mathtt{S_6}, \#\mathtt{S_7}) \\ &= (1, 0, 0, 0, 1, 2, 1)\end{aligned}$$

Zeit: Bei dem Transformationsmodus *Zeit* wird gezählt, wie lange jeder individuelle Sensor in einem Fall aktiviert ist:

$$\begin{aligned}\tau_1^T &= (t(\mathtt{S_1}), t(\mathtt{S_2}), t(\mathtt{S_3}), t(\mathtt{S_4}), t(\mathtt{S_5}), t(\mathtt{S_6}), t(\mathtt{S_7})) \\ &= (d_1, 0, 0, 0, d_4, d_3 + d_5, d_2)\end{aligned}$$

Anzahl & Zeit: Dieser Transformationsmodus verbindet den Zeit- und Anzahlvektor miteinander. Die beiden Vektoren werden einfach miteinander verkettet:

$$\begin{aligned}\tau_1^{QT} &= [\tau_1^Q,\ \tau_1^T] \\ &= (1, 0, 0, 0, 1, 2, 1,\ d_1, 0, 0, 0, d_4, d_3 + d_5, d_2)\end{aligned}$$

Wie die Transformation im Framework umgesetzt wurde, ist in Algorithmus 6 beschrieben. Im Folgenden sind die wichtigsten Aspekte des Algorithmus erklärt.

Algorithmus 6 Transformation Fälle zu Vektoren

```
1:  function TransformCasesToVectors(event_log_cases,
                                     vectorisation_method)
2:      if vectorisation_method ∈ {'quantity', 'quantity_time'} then
3:          for each case_i in event_log_cases do
4:              for each event_j in case_i do
5:                  k ← SensorID(event_j)
6:                  quantity_vector_ik ← quantity_vector_ik + 1
7:              end for
8:          end for
9:      end if
10:     if vectorisation_method ∈ {'time', 'quantity_time'} then
11:         for each case_i in event_log_cases do
12:             for each event_j in case_i do
13:                 k ← SensorID(event_j)
14:                 time_vector_ik ← time_vector_ik + Duration(event_j)
15:             end for
16:         end for
17:     end if
18:     if vectorisation_method = 'time' then
19:         return time_vector
20:     else if vectorisation_method = 'quantity' then
21:         return quantity_vector
22:     else if vectorisation_method = 'quantity_time' then
23:         quantity_time_vector ← concatenate(quantity_vector, time_vector)
24:         return quantity_time_vector
25:     else
26:         return None
27:     end if
28: end function
```

Zeile 1 – Funktionsdefinition: Definition der Funktion, die den Sensorereignislog (*event_log_cases*) und die Vektorisierungsmethode (*vectorisation_method*) übergeben bekommt.

Zeile 2 – Prüfung auf Quantity-Vektorisierung: Es wird überprüft, ob die ausgewählte Vektorisierungsmethode *quantity* oder *quantity_time* ist.

Zeile 3 & Zeile 11 – Iteration über Fälle: Iteriere über jeden einzelnen Fall (*case_i*), der im übergebenen Ereignislos enthalten ist.

Zeile 4 & Zeile 12 – Iteration über Ereignisse: Iteriere über jedes Ereignis (*event_j*) im aktuellen Fall (*case_i*).

Zeile 5 & Zeile 13 – Bestimmen der Sensor-ID: Extraktion der ID des Sensors, der bei dem aktuellen Ereignis aktiviert wurde. Diese ID wird temporär der Variable k zugewiesen.

Zeile 6 – Zählen der Ereignisse: Inkrementiere den Zähler für den aktuellen Sensor ($quantity\_vector_{ik}$) um eins, um die Anzahl der Ereignisse zu erfassen. Die Vektoren $quantity\_vector_i$ für alle i können als folgende Matrix dargestellt werden:

$$\begin{bmatrix} quantity\_vector_1 \\ quantity\_vector_2 \\ \vdots \\ quantity\_vector_i \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1k} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & x_{i3} & \dots & x_{ik} \end{bmatrix}$$

Die Laufvariable k steht für die fortlaufende Sensor-ID.

Zeile 14 – Summieren der Aktivierungszeit: Addiere die Dauer des aktuellen Ereignisses zur Gesamtdauer der Aktivierungen des betreffenden Sensors ($time\_vector_{ik}$).

$$\begin{bmatrix} time\_vector_1 \\ time\_vector_2 \\ \vdots \\ time\_vector_i \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1k} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & x_{i3} & \dots & x_{ik} \end{bmatrix}$$

Die Laufvariable k steht hier ebenfalls für die fortlaufende Sensor-ID.

Zeile 18 – Rückgabe des Zeitvektors: Wenn als Vektorisierungsmethode *time* gewählt wurde, wird der Zeitvektor (*time_vector*) zurückgegeben.

Zeile 21 – Rückgabe des Häufigkeitsvektors: Wenn *quantity* als Vektorisierungsmethode ausgewählt wurde, wird an dieser Stelle der Häufigkeitsvektor (*quantity_vector*) zurückgegeben.

Zeile 23 – Vektorverkettung (Quantity-Time): Wenn als Vektorisierungsmethode *quantity_time* ausgewählt wurde, wird an dieser Stelle der Häufigkeits- und Zeitvektor zu einem kombinierten Vektor (*quantity_time_vector*) verkettet.

Zeile 24 – Rückgabe des kombinierten Vektors: Ist für die Rückgabe die Kombination aus Häufigkeits- und Zeitvektor gewählt, wird an dieser Stelle der Vektor (*quantity_time_vector*) zurückgegeben.

Zeile 26 – Rückgabe bei ungültiger Methode: Falls eine Vektorisierungsmethode übergeben wurde, die (noch) nicht implementiert ist, wird der Wert `None` zurückgegeben.

Die vorgestellten Varianten der Transformation haben jedoch den Nachteil, dass die Reihenfolge der Sensoraktivierungen nicht berücksichtigt wird. So ergeben Bewegungen in exakt entgegengesetzte Richtungen den gleichen Vektor. In den Abbildungen 3.6a und 3.6d ist eine solche entgegengesetzte Bewegung dargestellt. Die Bewegung der Person im Raum wird durch eine stilisierte Figurendarstellung in den beiden Abbildungen visualisiert. Die Zahlen in den Figuren geben die zeitliche Komponente t an. Obwohl beide Szenarien unterschiedliche Ereignislogs erzeugen (Tabellen 3.6c und 3.6b) ist der resultierende aggregierte Vektor bei dem Transformationsmodus *Anzahl* der Gleiche:

$$\begin{aligned}\tau_N^Q = \tau_S^Q &= (\#\mathtt{M_1}, \#\mathtt{M_2}, \#\mathtt{M_3}, \#\mathtt{M_4}) \\ &= (1, 1, 1, 1)\end{aligned}$$

Sollte nun auch die Bewegungsgeschwindigkeit der Entitäten in den beiden Szenarien (Bewegungsrichtung nach Norden, Bewegungsrichtung nach Süden, Abbildungen 3.6a & 3.6d) gleich sein, so ergibt sich Folgendes:

$$(d_{N1} = d_{S4}) \wedge (d_{N2} = d_{S3}) \wedge (d_{N3} = d_{S2}) \wedge (d_{N4} = d_{S1})$$

Die Vektoren im Transformationsmodus *Zeit* und der zusammengesetzte Vektor aus *Zeit-* und *Anzahl* sind identisch. Dann sind die resultierenden Vektoren ebenfalls nicht zu unterscheiden. Für den Transformationsmodus **Zeit** ergibt sich folgender Vektor:

$$\begin{aligned}\tau_N^T = \tau_S^T &= (t(\mathtt{M_1}), t(\mathtt{M_2}), t(\mathtt{M_3}), t(\mathtt{M_4})))\\ &= (d_1, 0, 0, 0, d_4, d_3 + d_5, d_2)\end{aligned}$$

Bei der Verkettung der beiden Vektoren (**Anzahl & Zeit**) ergibt sich natürlicherweise auch ein identischer Vektor für beide Bewegungsrichtungen:

$$\begin{aligned}\tau_N^{QT} = \tau_S^{QT} &= [\tau_N^Q,\, \tau_N^T] = [\tau_S^Q,\, \tau_S^T]\\ &= (1, 0, 0, 0, 1, 2, 1, d_1, 0, 0, 0, d_4, d_3 + d_5, d_2)\end{aligned}$$

Diese Grenzfälle, insbesondere unter Einbezug der Aktivierungszeiten, kommen in der Praxis üblicherweise nicht häufig vor. Können jedoch bei besonderen Domänen mit speziellem (Sensor)layout (beispielsweise lange enge Gänge und sich gleichförmig schnell bewegende Entitäten) sehr wohl vermehrt vorkommen.

Clustering-Algorithmus anwenden

Die im Schritt zuvor vektorisierten Fälle werden nun in Gruppen entsprechend ähnlicher Fälle zusammengefasst. Das Ziel hierbei ist es, für jedes Cluster ein repräsentatives Prozessmodell zu finden. Da es sich um Sensordaten handelt, die komplexe Abläufe, zum Beispiel menschliche Bewegungsmuster, aufzeichnen, ist es eher unwahrscheinlich, dass exakt gleiche Fälle auftreten. Somit wäre es nicht effizient, für jede Fallvariation ein eigenes Cluster zu bilden und auf ein Prozessmodell abzubilden. Die vorgeschlagene Lösung dafür ist, leichte Schwankungen im Ablauf der einzelnen Fälle als ähnlich genug für die gleiche Clusterzuordnung zuzulassen.

In diesem Schritt können zwei Arten von Rauschen auftreten. Erstens könnten Fälle unpassenden Clustern zugeordnet worden sein. Zweitens könnten Fälle einem geeigneten Cluster zugeordnet worden sein, enthalten jedoch Sensorwerte, die nicht repräsentativ für diesen Cluster sind. Um die fehlerhaft klassifizierten Fälle und

(a) Bewegung Richtung Norden

Zeit	Sensor	Aktivierungsdauer
t_1	M_{01}	d_{N1}
t_2	M_{02}	d_{N2}
t_3	M_{03}	d_{N3}
t_4	M_{04}	d_{N4}

(b) Sensorereignislog zu Abbildung 3.6a

Zeit	Sensor	Aktivierungsdauer
t_1	M_{04}	d_{S1}
t_2	M_{03}	d_{S2}
t_3	M_{02}	d_{S3}
t_4	M_{01}	d_{S4}

(c) Sensorereignislog zu Abbildung 3.6d

(d) Bewegung Richtung Süden

Abbildung 3.6: Sensorereignislog

falsch aufgezeichneten Sensoren zu filtern, werden die Häufigkeiten der Sensoraktivierungen innerhalb eines Clusters verglichen. Sensoren, die seltener aktiviert werden, werden als Rauschen klassifiziert. In der Implementierung des Frameworks wurde ein Schwellenwert integriert, der es ermöglicht, das Niveau des Rauschfilters anzupassen.

Das Ergebnis dieses Schrittes ist eine Menge von Clustern, die jeweils eine Aktivität repräsentieren. Jede Aktivität $a \in A$ hat sowohl ein Aktivitätslabel $label(a)$ als auch ein Prozessmodell, das das Low-Level-Verhalten dieser Aktivität a in Form von Ereignissen auf der Sensorebene beschreibt.

Die Aktivitätslabel der Cluster bestehen vorerst nur aus Pseudo-Label (Clustern 1, 2, 3, ..., n). Dies reduziert den manuellen Beschriftungsaufwand und erhöht die Label-Qualität, wenn Cluster im Kontext betrachtet werden können. Erst nach Abschluss der Hyperparameter-Optimierung ersetzt der Domänenexperte die Pseudo-Label durch konkrete Prozessaktivitätsbeschriftungen für das Prozessmodell mit der höchsten Qualität.

Nach der Entdeckung von Clustern ähnlicher Fälle, die unterschiedlichen Aktivitäten entsprechen, wurde noch nichts zur Semantik der Aktivitäten ausgesagt, die ein Cluster repräsentiert. Außerdem kann es schwierig sein, die Qualität der Cluster zu beurteilen. Es ist davon auszugehen, dass für das Labeln (Beschriften) von Aktivitäten im Allgemeinen ein Mensch mit entsprechendem Fachwissen erforderlich ist.

Die gewählte Vorgehensweise hat drei Vorteile:

Kontextbasierte Beschriftung: Cluster werden nicht isoliert beschriftet, sondern im Kontext vorheriger und nachfolgender Cluster bzw. Prozessaktivitäten.

Einmalige Beschriftung: Beschriftung erfolgt nur einmal, nämlich für einen entdeckten Prozess hoher Qualität.

Gezielte Inspektion: Bei doppelten Pseudo-Labels können durch eine Inspektion der Cluster, basierend auf der zugrunde liegenden Sensoraktivierung weitere Erkenntnisse gewonnen werden. Beispielsweise, dass zwei Cluster nicht identisch sind, sondern sich auf unterschiedliche Aktivitäten beziehen.

Eine (halb)automatisierte Beschriftung der Cluster reduziert den Aufwand für manuelle Beschriftungen insgesamt deutlich.

Um die Ergebnisse des Clusterings interpretierbar zu machen, wird im Nachfolgenden für jedes Cluster ein Prozessmodell entdeckt.

Clustering mit *k*-Means

Das Clustern der Traces mit dem k-Means Verfahren erfolgt mit dem Python Paket *KMeans* aus dem scikit-learn Projekt (Pedregosa et al. 2011). Die theoretischen Grundlagen zum k-Means Verfahren sind in 2.7 beschrieben. Die Einbindung des k-Means Algorithmus im Framework dient in erster Linie als Baseline-Vergleich, um zu überprüfen, ob die Wahl eines komplexeren Clustering-Algorithmus effizienter ist.

Die Anzahl der Cluster ist auch als Teil der Hyperparameteroptimierung variabel und je nach Anwendungsdomäne ergibt sich ein anderer Wert für die Clusteranzahl k.

Clustering mit k-Medoids

Der Vorteil des k-Medoids-Algorithmus gegenüber dem k-Means-Verfahren ist, dass die Schwerpunkte der Cluster (Centroids) Vektoren sind, die auch tatsächlich im Datensatz auftreten. Die Clusterschwerpunkte sind also aussagekräftige und interpretierbare Vektoren.

Für die Implementierung wurde der *KMeans* Algorithmus aus dem *sklearn_extra* Paket vom scikit-learn Projekt (Pedregosa et al. 2011) verwendet.

Für das k-Medoids-Clustering gibt es eine benutzerdefinierte Distanzfunktion, sodass man nicht auf eine Vektortransformation angewiesen ist. Anstatt des euklidischen Abstands zwischen Vektoren zu berechnen, verwendet der angepasste k-Medoids-Ansatz die Distanzmatrix des Smart Homes und die tatsächlichen Abstände der Sensoren im Smart Home, um ähnliche Fälle zu gruppieren.

Clustering mit Self-Organising Maps

Die Self-Organizing Map (SOM) (Kohonennetz) wird zu Beginn mit festgelegten Dimensionen und Lernparametern initialisiert. Die Parameter m und n definieren dabei die Größe der neuronalen Karte, wobei m die Anzahl der Knoten in vertikaler Richtung (Zeilen) und n die Anzahl der Knoten in horizontaler Richtung (Spalten) angibt. Der Parameter *dim* bestimmt die Größe des Eingabevektors, der der Anzahl der Merkmale (Dimensionen) in den Trainingsdaten entspricht. Die Lernrate (*learning_rate*) definiert, mit welcher Schrittgröße die Gewichte der SOM während des Trainings aktualisiert werden. Der in dieser Arbeit verwendete Trainingsalgorithmus ist in Algorithmus 7 dargestellt und verwendet die Implementierung aus dem Python-Paket *sklearn_som*[1].

Nach der Initialisierung wird die SOM mit zu Vektoren transformierten Fällen aus dem Ereignisprotokoll (*event_log_cases*) trainiert (Algorithmus 7, Zeile 3). Während dieses Trainingsprozesses wird die neuronale Karte iterativ an die Eingabedaten angepasst. Ziel dieses Anpassungsvorgangs ist es, Muster in den Fällen zu erkennen und dadurch ähnliche Fälle in Clustern zusammenzufassen.

[1] `https://pypi.org/project/sklearn-som/`

Algorithmus 7 Training der Self-Organizing Map (SOM)

```
1: function CreateSomWithSklearn(m, n, dim, learning_rate,
                                 event_log_cases)
2:     som ← SOM(m, n, dim, learning_rate)
3:     Fit(som, event_log_cases.values)              ▷ Train SOM with data
4:     if som.inertia < target.inertia then
5:         clustering_result ← Predict(som, event_log_cases.values)
6:         inertia ← som.inertia
7:     end if
8:     return clustering_result, inertia
9: end function
```

Zur Bewertung der Qualität der entstandenen Cluster wird die sogenannte Trägheit (*inertia*) der SOM verwendet. Diese Trägheit beschreibt, wie die SOM die Daten repräsentiert: Eine niedrigere Trägheit bedeutet eine bessere Anpassung der Karte an die Daten. Während des Prozesses der Vorhersage (`PREDICT`: Algorithmus 7, Zeile 5) wird überprüft, ob die Trägheit der aktuellen SOM geringer ist als die bisher beste gefundene Trägheit. Ist dies der Fall, wird das aktuelle Modell übernommen und dessen Clustering-Ergebnis als das derzeit beste Modell gespeichert.

Schließlich liefert die in Algorithmus 7 dargestellte Funktion zwei Parameter zurück: die Trägheit des besten SOM-Modells als Maß für die Qualität der Datengruppierung und die zugehörige Einteilung der Cluster der betrachteten Fälle.

3.4 Process Discovery

Die Entdeckungen von Prozessen (Process Discovery) finden im Framework an zwei Stellen statt:

Process Discovery der Cluster: In jedem Cluster werden mehrere ähnliche Fälle zusammengefasst. Um für jedes der Cluster eine aussagekräftige Bezeichnung der Aktivitäten ermitteln zu können, werden mithilfe von pm4py sogenannte Directly-Follows-Graphen (DFG) erzeugt. Diese stellen die typischen Abfolgen von Sensoraktivierungen innerhalb der Cluster dar. Mithilfe eines Filters werden selten auftretende Aktivitäten herausgefiltert, um so übersichtliche und interpretierbare Prozessmodelle zu erhalten.

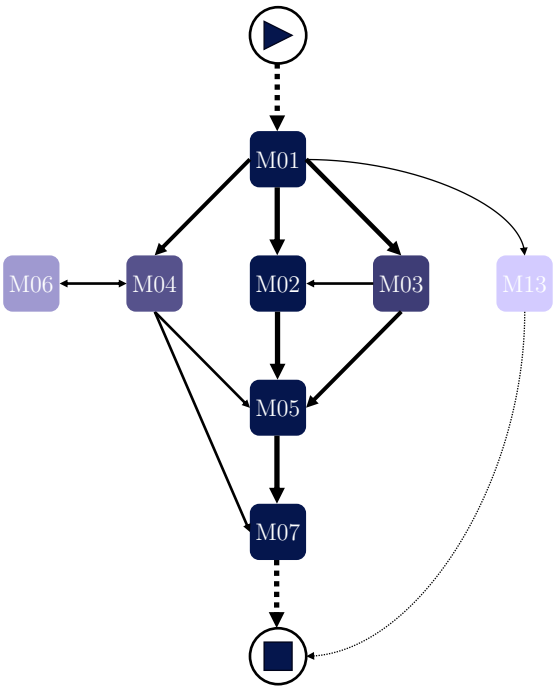


Abbildung 3.7: Directly-Follows Graph eines Beispielclusters

Process Discovery der Tagesabläufe: In diesem Teil wird versucht, typische Prozessmodelle abzuleiten, die tägliche Routinen widerspiegeln. Da menschliche Tagesabläufe, speziell im privaten Umfeld, viele Unregelmäßigkeiten aufweisen können, werden die Tage in kleinere Zeitabschnitte (sogenannte Routinen) sowie nach Wochentagen oder Wochenende aufgeteilt. Die Idee ist, dadurch übersichtlichere und interpretierbare Prozessmodelle zu erhalten.

Die Vorgehensweise der beiden Process Discovery Schritte ist in den folgenden beiden Unterkapiteln beschrieben.

Process Discovery: Cluster

In jedem der gefundenen Cluster sind nun mehrere Fälle zusammengefasst. Abbildung 3.7 zeigt beispielhaft das Ergebnis des Process Discovery Schrittes der Cluster. In Abbildung 3.7 ist ein Directly-Follows Graph dargestellt, der die Sensoraktivitäten des Beispielclusters und deren direkte Folgebeziehung beinhaltet. Durch die Farbabstufung und die Pfeilstärke sind Häufigkeitsbeziehungen ebenfalls enthalten.

Das Ziel ist es, für jedes Cluster eine Aktivitätsbezeichnung zu finden, die die enthaltenen Fälle zusammenfassend am geeignetsten beschreibt. Um dies bestmöglich zu

erreichen, wird für jedes Cluster mithilfe von pm4py (Berti et al. 2023) ein Directly-Follows Graph (DFG) erstellt. Die Vorgehensweise ist in Algorithmus 8 beschrieben.

Zeile 2 – Aktive Sensoren: Insbesondere beim Betrachten von Sensoren, welche durch Bewegungen aktiviert werden, ist es bei einem Directly-Follows Graphen sinnvoll, sich auf die aktivierten Sensoren zu beschränken. Aus diesem Grund werden alle Ereignisse, bei denen Sensoren deaktiviert wurden, aus dem Ereignislog entfernt.

Zeile 3 – Eindeutige Liste der Cluster: Um im nächsten Schritt über jeden der Cluster iterieren zu können, muss zuerst eine Liste erstellt werden, welche jeden der Cluster eindeutig identifiziert.

Zeile 4 – For-Schleife: In der For-Schleife wird über alle Cluster iteriert, um für jeden Cluster ein Prozessmodell zu erstellen.

Zeile 5 – Häufigkeitsschwelle bestimmen: Um die zu entdeckenden Directly-Follows-Grafen nicht durch selten auftretende Sensoren unnötig zu verkomplizieren, wird ein Filter eingeführt. Hierbei wird ein relativer Schwellwert als Parameter übergeben, der für alle entdeckten DFG in der aktuellen Hyperparameteroptimierungsiteration gilt. Dieser wird in Zeile 5 in einen absoluten ganzzahligen Wert umgerechnet. Der absolute Wert richtet sich nach der Häufigkeit des am häufigsten vorkommenden Sensors.

Zeile 6 – Filter Anwenden: Das zuvor festgelegte Filterkriterium wird angewendet und alle selten auftretenden Sensoraktivierungen werden entfernt.

Zeile 7 & 8 – Graph erstellen: Mithilfe von pm4py (Berti et al. 2023) wird an dieser Stelle ein Directly-Follows Graph erstellt und als Grafik zur späteren Analyse exportiert.

Process Discovery: Tagesabläufe

Um die Prozessmodelle der Tagesabläufe zu entdecken, sind zwei Schritte notwendig:

Routinen erstellen: Die Entdeckung ganzer Tagesabläufe in Ereignisdaten führt zu komplexen und schwer interpretierbaren Prozessmodellen, da menschliches Verhalten stark von Unregelmäßigkeiten geprägt ist. Um diesem Problem zu begegnen, wird der Tag in kleinere, logisch sinnvolle Abschnitte unterteilt. Dies

Algorithmus 8 Process Discovery: Cluster

```
1: procedure CreateActivityModels(clustering_results,
                                  relative_dfg_threshold)
2:     Retain only active sensor events from clustering_results
3:     clusters ← unique clusters extracted from clustering_results
4:     for each cluster in clusters do
5:         min_number_of_occurrences ←
             ⌊1/2 + (max(#of activites in cluster) × relative_dfg_threshold)⌋
6:         Filter activities in cluster to retain only those with
             occurrences ≥ min_number_of_occurrences
7:         Generate a DFG from filtered activities in cluster
8:         Export the generated DFG as an image file
9:     end for
10: end procedure
```

kann nach Tageszeit oder Wochentag erfolgen. Das Framework erlaubt dabei verschiedene Methoden der Aufteilung: ohne Trennung, nach Arbeitstag/Wochenende oder nach einzelnen Wochentagen. Zusätzlich kann der Tagesverlauf in zwei bis fünf Routinen unterteilt werden (z.B. „Morgen“, „Nachmittag“, „Abend“), um typische Verhaltensmuster besser darstellen zu können. Auf Basis dieser Segmentierungen werden differenzierte Prozessmodelle erzeugt, die interpretierbare Routinen innerhalb homogener Tagesabschnitte abbilden.

Prozessmodelle entdecken: Aus den zuvor gruppierten Ereignisdaten werden mithilfe von Process Mining-Techniken automatisiert Prozessmodelle erzeugt. Für jede Routine wird ein Ereignislog extrahiert, in ein standardisiertes Format (XES) überführt und daraus ein Directly-Follows Graph sowie ein Petri-Netz generiert. Die Güte der entdeckten Modelle wird über Metriken wie Fitness, Precision oder den daraus berechneten F_1-Score bewertet. Die erzeugten Prozessmodelle lassen sich grafisch darstellen und können von Domänenexperten manuell validiert und mit sinnvollen Aktivitätsbezeichnungen versehen werden. Ziel ist es, interpretierbare und vergleichbare Routinen abzuleiten, die ausschließlich auf Sensordaten basieren und zur Verhaltensanalyse im Alltag beitragen.

Die Einführung von *Routinen* (siehe Abb. 3.8b) hat das Ziel, ein Prozessmodell für jedes Segment des Tages/der Woche anstelle eines einzigen Prozessmodells für den gesamten Tag zu entdecken. Bei der Berechnung des gesamten F_1-Scores zeigt sich,

dass eine einfache Zusammenfassung der verschiedenen F_1-Scores pro Prozessmodell nicht ohne Weiteres möglich ist, da die Anzahl der Ereignisse je nach Tag/Woche unterschiedlich ist. Daher wird ein gewichteter F_1-Score für eine ausbalancierte Berechnung verwendet.

Im Process Discovery Schritt für die Tagesabläufe wird versucht, aus den in den vorigen Schritten vorverarbeiteten Sensordaten ein Prozessmodell zu entdecken.

Routinen erstellen

Experimente im Rahmen der Publikation aus Anhang C (Janssen et al. 2020) haben jedoch gezeigt, dass es beim Entdecken ganzer Tagesabläufe zu schwer interpretierbaren Prozessmodellen kommt. Eine Erklärung hierfür ist, dass menschliche Aktivitäten im privaten Kontext mit zu vielen Unregelmäßigkeiten behaftet sind. Die Abläufe an verschiedenen Tagen unterscheiden sich zu stark, um ein allgemein gültiges Prozessmodell entdecken zu können, welches die Abläufe an verschiedenen Tagen zusammenfasst. Aus diesem Grund ist es sinnvoll, den gesamten Tag in mehrere Abschnitte zu unterteilen und nicht jeden Tag gleich zu behandeln. Dies ermöglicht, dass in bestimmten Tagesabschnitten eine beobachtbare Routine zu entdecken ist, bzw. dass an gleichen Wochentagen ähnliche Routinen auftreten. Aus diesem Grund wird nun nicht mehr ein einziges Prozessmodell für einen repräsentativen Tag von morgens bis nachts erstellt, sondern je nach Unterteilung werden mehrere Prozessmodelle erstellt.

Folgende Unterteilungen der Routinen wurden im Framework betrachtet:

Wochenaufteilung: Die Aufteilung der Tage ist in Algorithmus 9 beschrieben. Die Aufteilung, die die besten Ergebnisse liefert, hängt stark von der Anwendungsdomäne ab. Bei gewissen Anwendungsdomänen kann es sein, dass es Tagesroutinen gibt, die nicht durch die drei unten vorgeschlagenen Aufteilungen abgebildet werden können. In diesem Fall müsste eine neue benutzerdefinierte Aufteilung der Implementierung hinzugefügt werden. Die Implementierung ist als Pseudocode in Algorithmus 9 beschrieben. Das Framework unterstützt in der aktuellen Version drei Aufteilungen der Tage. Die Unterscheidung der Fälle erfolgt in den Zeilen 2, 10 und 14 im Algorithmus 9. Die verschiedenen Aufteilungen sind nachfolgend beschrieben:

Algorithmus 9 Routine hinzufügen

```
1: function AddRoutine(clustering_results, week_separator, day_partitions)
2:     if week_separator = 'workday' then
3:         for each event in event_log_cluster do:
4:             if day of the week in Timestamp < 5 then
5:                 'Routine' in event ← 'weekday'
6:             else
7:                 'Routine' in event ← 'weekend'
8:             end if
9:         end for
10:    else if week_separator = 'weekday' then
11:        for each event in event_log_cluster do:
12:            'Routine' in event ← corresponding weekday name derived
                                    from Timestamp
13:        end for
14:    else
15:        for each event in event_log_cluster do:
16:            'Routine' in event ← None
17:        end for
18:    end if
19:    event_log_cluster[DayPartition] ← PartitionDay(clustering_results,
                                                       day_partitions)
20:    return event_log_cluster
21: end function
```

Arbeitstag/Wochenende: Bei dem Unterteilungsmodus zwischen Arbeitstagen (Montag – Freitag) und Wochenenden (Samstag, Sonntag) wird davon ausgegangen, dass sich unter der Woche Muster täglich wiederholen, die Aktivitäten an Wochenenden aber fundamental andere Muster aufweisen. Wenn der Parameter hyp_week_separator auf "workday"gesetzt ist, werden Tage als 'weekday' (Wochentag) oder 'weekend' (Wochenende) klassifiziert. Dies geschieht basierend darauf, ob der Tag der Woche (Montag bis Freitag) unterhalb von 5 liegt.

Wochentag: Die Unterteilung nach Wochentagen unterscheidet zwischen jedem einzelnen Wochentag (Montag, Dienstag, Mittwoch, ...). Bei dieser Art der Unterteilung wird davon ausgegangen, dass das Verhalten an keinem Wochentag dem anderen gleicht; identische Wochentage aber

ähnliches Verhalten aufweisen. Wie in Algorithmus 9 in Zeile 2 beschrieben.

Keine: Wenn keine Unterteilung angewendet wird, wird jeder Tag gleich behandelt (Algorithmus 9, Zeile 16). Es gibt keine Unterteilung nach unterschiedlichen Wochentagen bzw. Wochenenden. Dies eignet sich für Domänen, bei denen es täglich wiederholende Muster gibt, die unabhängig von Wochentagen immer ähnlich auftreten.

Tagesaufteilung: Bei der Partitionierung des Tages wird ein Tag weiter in kleinere Teile unterteilt, die wiederholende Routinen widerspiegeln, z.B. morgendliche Routinen oder abendliche Ruheroutinen im Privathauskontext. Die Anzahl der Unterteilungen wirkt sich auf die entdeckten Prozesse aus: Eine morgendliche Routine stellt einen anderen übersichtlichen Prozess dar als eine für den ganzen Tag betrachtete Routine. Abhängig von der Variable *day_partitions* wird in Algorithmus 9, Zeile 19, der Tag weiter in spezifische Abschnitte durch den Aufruf der Methode beschrieben in Algorithmus 10, unterteilt:

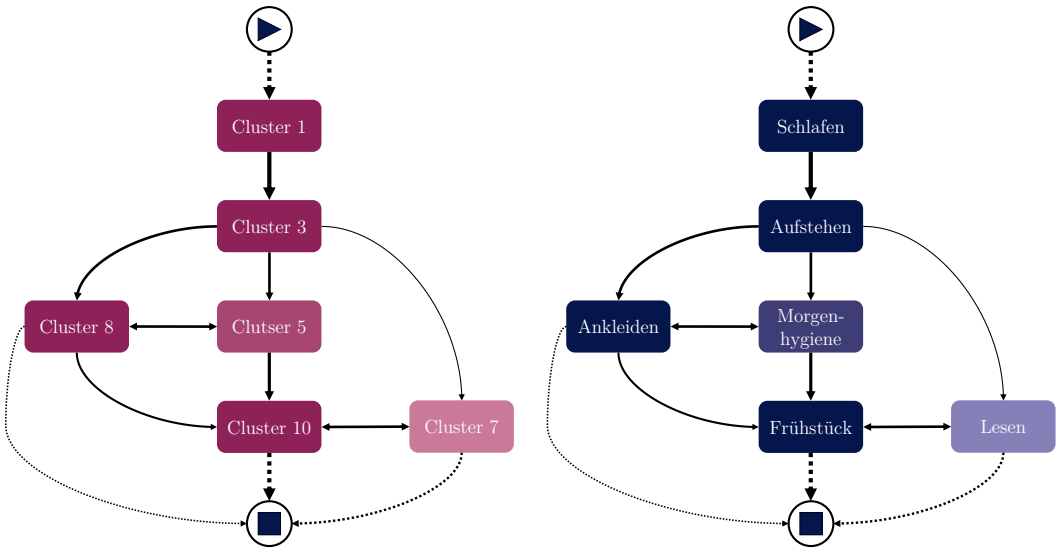
- 2 Abschnitte: Vormittag (AM) und Nachmittag (PM).
- 3 Abschnitte: Nacht, Tag und Abend.
- 4 Abschnitte: Nacht, Morgen, Nachmittag und Abend.
- 5 Abschnitte: Nacht, Morgen, Mittag, Nachmittag und Abend.

Algorithmus 10 Einteilung der Tage

```
1: function PartitionDay(event_log_cluster, day_partitions)
2:     event_log_cluster_routine ← event_log_cluster
3:     for each event in event_log_cluster_routine do
4:         if day_partitions = 1 then
5:             'Routine' in event ← None
6:         else if day_partitions = 2 then
7:             if hour of Timestamp in event ∈ [0, 11] then
8:                 'Routine' in event ← 'AM'
9:             else
10:                'Routine' in event ← 'PM'
11:            end if
```

Algorithmus 10 Einteilung der Tage (Fortsetzung)

```
12:         else if day_partitions = 3 then
13:             if hour of Timestamp in event ∈ [0, 8] then
14:                 'Routine' in event ← 'night'
15:             else if hour of Timestamp in event ∈ [9, 16] then
16:                 'Routine' in event ← 'day'
17:             else
18:                 'Routine' in event ← 'evening'
19:             end if
20:         else if day_partitions = 4 then
21:             if hour of Timestamp in event ∈ [0, 6] then
22:                 'Routine' in event ← 'night'
23:             else if hour of Timestamp in event ∈ [7, 12] then
24:                 'Routine' in event ← 'morning'
25:             else if hour of Timestamp in event ∈ [13, 18] then
26:                 'Routine' in event ← 'afternoon'
27:             else
28:                 'Routine' in event ← 'evening'
29:             end if
30:         else if day_partitions = 5 then
31:             if hour of Timestamp in event ∈ [6, 10] then
32:                 'Routine' in event ← 'morning'
33:             else if hour of Timestamp in event ∈ [11, 14] then
34:                 'Routine' in event ← 'noon'
35:             else if hour of Timestamp in event ∈ [15, 17] then
36:                 'Routine' in event ← 'afternoon'
37:             else if hour of Timestamp in event ∈ [18, 23] then
38:                 'Routine' in event ← 'evening'
39:             else
40:                 'Routine' in event ← 'night'
41:             end if
42:         end if
43:     end for
44:     return event_log_cluster_routine
45: end function
```



(a) Platzhalter für Aktivitätsbezeichnungen

(b) Aktivitätsbezeichnungen

Abbildung 3.8: Beispiel für einen Directly-Follows Graph einer Morgenroutine.

Prozessmodelle entdecken

In Abbildung 3.8 sind zwei Directly-Follows Graphen dargestellt, wie sie in dem Verarbeitungsschritt in diesem Abschnitt neben Petri-Netzen der Routinen entdeckt werden sollen. Die rechte Abbildung 3.8a zeigt ein Beispiel für ein Prozessmodell, bei dem die Aktivitätsbezeichnungen noch mit Platzhalternamen versehen sind. Die Darstellung im linken Teil der Abbildung (3.8b) mit den Aktivitätsbezeichnungen ist das Ergebnis, wenn die Platzhalternamen durch einen Domänenexperten manuell ersetzt wurden.

Die Vorgehensweise ist in Algorithmus 11 beschrieben und im Folgenden werden die wichtigen Teile des Codes erklärt:

Zeile 1 – Funktionsdefinition: Die Funktion `CreateProcessModel` erhält den Eingabeparameter `output_case_traces_cluster`, eine strukturierte Sammlung von Ereignislogs, gruppiert nach Routinen.

Algorithmus 11 Create Process Model from Event Log Data

```
1: function CreateProcessModel(output_case_traces_cluster)
2:     Remove inactive entries from output_case_traces_cluster
3:     routine_list ← unique routines extracted from output_case_traces_cluster
4:     i ← 0
5:     events_per_routine ← Initialize empty data frame
6:     routine_metric ← Initialize empty data frame
7:     for each routine in routine_list do
8:         routine_log ← Filter event log (output_case_traces_cluster)
                         by routine of current iteration
9:         Convert filtered event logs into a format compatible with
             process mining tools
10:        Export the formatted event logs as XES files
11:        Generate and export Directly-Follows Graph (DFG) as an image file
12:        events_per_routine_i ← Count the number of events for this routine
13:        routine_metric_i ← ApplyMiner(routine_log, miner_type
                                         metric_to_be_maximised)
14:        i ← i+1
15:    end for
16:    weighted_metric_average ← (Σ_{i=1}^{n} (events_per_routine_i × routine_metric_i)) / (Σ_{i=1}^{n} events_per_routine_i)
17:    return weighted_metric_average
18: end function
```

Zeile 2 – Entfernen inaktiver Einträge: Entferne alle inaktiven oder irrelevanten Einträge aus dem Ereignislog, sodass nur relevante Routinen für die Modellierung übrig bleiben. Das Deaktivieren eines Sensors bedarf keines Eintrags in der grafischen Prozessdarstellung.

Zeile 3 – Liste von Routinen extrahieren: Extrahiere alle einzigartigen Routinen aus dem bereinigten Ereignislog. Diese Routinen bilden die Grundlage für die Schleife in Zeile 7.

Zeile 4 – Initialisierung Laufindex: Setze einen Laufindex `i` auf 0. Dieser wird verwendet, um Metriken und die Anzahl an Ereignissen je Routine zu speichern.

Zeile 5 & Zeile 6 – Datenstruktur initalisieren: Initialisiere zwei leere Datenstrukturen: `events_per_routine` zur Speicherung der Anzahl von Ereignissen je Routine und `routine_metric` zur Speicherung der resultierenden Metriken pro Routine.

Zeile 7 – for-Schleife Routinen: Für jede Routine wird ein Directly-Follows Graph erstellt und die Miner-Methode aufgerufen, um ein Petri-Netz zu entdecken.

Zeile 8 – Filtern nach Routine: Filtere das Ereignislog, sodass nur die Einträge enthalten sind, die zur aktuellen Routine gehören.

Zeile 9 – Formatumwandlung: Wandle das gefilterte Ereignislog in ein Format um, das von Process Mining-Tools (Pm4Py) verarbeitet werden kann. Insbesondere die Spaltenbezeichnungen müssen mit Pm4Py kompatibel sein.

Zeile 10 – Export als XES: Speichere das Ereignislog im standardisierten XES-Format, um eine Kompatibilität mit weiteren Tools zu gewährleisten.

Zeile 11 – Export DFG: Erzeuge ein Bild des Directly-Follows Graphs (DFG) für die aktuelle Routine und speichere es im angegebenen Verzeichnis. Die Darstellung des exportierten DFGs entspricht ungefähr der Darstellung in Abbildung 3.8a

Zeile 12 – Ereigniszählung: Die Anzahl der Ereignisse in der aktuellen Routine wird gezählt und in der Liste `events_per_routine` an der Position `i` gespeichert.

Zeile 13 – Process Mining Algorithmus Aufruf: Aufruf der Miner-Methode (Algorithmus 12). Berechnung von Metriken mithilfe von Process Mining Werkzeugen und Speicherung der Ergebnisse.

Zeile 14 – Inkrementieren des Index: Der Laufindex `i` wird um den Wert 1 für die nächste Iteration erhöht.

Zeile 16 – Gewichtetes arithmetisches Mittel: Berechne den gewichteten Mittelwert für jede Metrik in der Menge der Routine-Metriken gemäß folgender Formel:

$$\frac{\sum_{i=1}^{n}(events\_per\_routine_i \times routine\_metric_i)}{\sum_{i=1}^{n} events\_per\_routine_i}$$

Dabei bezeichnet i den Laufindex und n die Anzahl der Routinen.

Zeile 17 – Rückgabe: Gib den gewichteten Mittelwert der Metriken zurück. Diese aggregierte Kennzahl kann als zusammenfassende Gesamteinschätzung der Prozessmodelle aller Routinen interpretiert werden.

Der Methodenaufruf aus Algorithmus 11 Zeile 13 ruft eine Methode auf, die für das Entdecken der Prozessmodelle in Form von Petri-Netzen zuständig ist und für das Errechnen der Qualitätsmaße zur Bewertung der entdeckten Prozessmodelle. Die Methode ist in Algorithmus 12 beschrieben. Basierend auf dem angegebenen *miner_type* wird festgelegt, welcher Process Mining Algorithmus angewendet wird. Zur Auswahl stehen der Heuristic und Inductive Miner. Der ausgewählte Process Mining Algorithmus wird auf das erstellte Ereignislog angewendet, um ein Prozessmodell in Form eines Petri-Netzes zu erzeugen. Das erzeugte Petri-Netz wird sowohl als Bild (PNG) als auch in einem standardisierten PNML-Format exportiert. Je nachdem, welche Metrik als Zielfunktionswert zur Maximierung ausgewählt wurde (*metric_to_be_maximised*), wird die entsprechende Metrik berechnet. Mögliche Metriken sind Fitness, Precision oder der F_1-Score. Die wichtigen Schritte des Pseudocodes sind im Folgenden kurz beschrieben.

Zeile 1 – Funktionsdefinition: Die Funktion `ApplyMiner` bekommt die folgenden Variablen übergeben: Den Ereignislog (*routine_log*), den ausgewählten Process Mining Algorithmus (*miner_type*) und die zu maximierende Metrik (*metric_to_be_maximised*).

Zeile 3 – Anwendung des Heuristic Miners: Der Heuristic Miner wird auf den übergebenen Ereignislog angewendet. In der Variable *petri_net* wird das resultierende Petri-Netz gespeichert.

Zeile 5 – Anwendung des Inductive Miners: Anwendung des Inductive Miners auf den Ereignislog. Das erzeugte Petri-Netz wird in der Variable *petri_net* gespeichert.

Zeile 7 – Kein gültiger Minertyp: Die Ausführung wird abgebrochen und der Wert `None` zurückgegeben, falls kein gültiger Minertyp übergeben wurde.

Zeile 9 – Export des Petri-Netzes als PNG: Das erzeugte Petri-Netz wird als PNG-Grafikdatei exportiert.

Algorithmus 12 Apply Miner and Compute Metrics

```
1: function ApplyMiner(routine_log, miner_type, metric_to_be_maximised)
2:     if miner_type = 'heuristic' then
3:         petri_net ← HeuristicMiner(routine_log)
4:     else if miner_type = 'inductive' then
5:         petri_net ← InductiveMiner(routine_log)
6:     else
7:         return None
8:     end if
9:     ExportAsPNG(petri_net)
10:    ExportAsPNML(petri_net)
11:    if metric_to_be_maximised = 'Fitness' then
12:        metric ← ComputeFitness(petri_net, routine_log)
13:    else if metric_to_be_maximised = 'Precision' then
14:        metric ← ComputePrecision(petri_net, routine_log)
15:    else if metric_to_be_maximised = 'F1' then
16:        fitness ← ComputeFitness(petri_net, routine_log)
17:        precision ← ComputePrecision(petri_net, routine_log)
18:        metric ← 2 / (fitness^-1 + precision^-1)
19:    end if
20:    return metric
21: end function
```

Zeile 10 – Export des Petri-Netzes als PNML: Das erzeugte Petri-Netz wird außerdem im standardisierten PNML-Dateiformat gespeichert; dies ermöglicht die Weiterverarbeitung in anderen Process Mining Werkzeugen.

Zeile 12 – Berechnung der Fitness: Die tokenbasierte Fitness-Metrik wird anhand des Ereignislogs und des erzeugten Petri-Netzes berechnet und in der Variable *metric* gespeichert.

Zeile 14 – Berechnung der Precision: Die tokenbasierte Precision-Metrik des Petri-Netzes in Bezug auf das Ereignislog wird berechnet und in der Variable *metric* gespeichert.

Zeile 18 – Berechnung des F_1-Scores: Berechnung des F_1-Scores, der als harmonisches Mittel aus Fitness und Precision wie folgt bestimmt wird:

$$F_1\text{-Score} = \frac{2}{fitness^{-1} + precision^{-1}}$$

Der Wert wird in der Variable *metric* gespeichert.

Zeile 20 – Rückgabe der Metrik: Die berechnete Metrik wird zurückgegeben, um sie zur Bewertung der Qualität des erstellten Prozessmodells zu verwenden.

In Algorithmus 12 wird ein Prozessmodell auf der Grundlage der Ereignisse jedes Clusters mit Hilfe des Inductive Miners erstellt. Die Qualität des Prozessmodells wird auf der Grundlage des F_1-Scores bewertet, der sich aus der Kombination der gemeinsamen Maße *Fitness* und *Precision* ergibt. Diese interpretierbaren Prozessmodelle machen die Methode für komplexe Prozesse, z.B. unstrukturierte Prozesse geeignet, und das Qualitätsmaß kann zur Validierung des Clustering-Ergebnisses verwendet werden. Mit dem Zugang zu den Prozessmodellen und ihrer Qualitätsbewertung kann ein Domänenexperte jedes Cluster interpretieren, validieren und mit einem geeigneten Aktivitätslabel kennzeichnen. Als Voraussetzung für das Entdecken aussagekräftiger Prozessmodelle wird die Annahme zugrunde gelegt, dass regelmäßiges und routinemäßiges Verhalten beobachtbar ist. Als Ausgangspunkt muss eine Aktivität ausgewählt werden, die höchstwahrscheinlich der Ursprung des Routineverhaltens ist, wie z. B. das Betreten des beobachteten Bereichs. Schließlich wird mit einer Standardtechnik, z. B. Inductive Miner nach Leemans (2018) ein Gesamtprozessmodell ermittelt. Die Ausgabe ist ein Prozessmodell, das das beobachtete Verhalten der Entitäten widerspiegelt, die nur aus den Sensor-Rohdaten aggregiert wurden.

3.5 Hyperparameteroptimierung

Um die Übertragbarkeit des Ansatzes auf andere Bereiche, wie z.B. intelligente Fabriken oder vernetzte Städte, zu gewährleisten, muss sichergestellt werden, dass die effizienten Parameter für jedes Szenario unabhängig voneinander gefunden werden können. Der spezifische Bereich der Parameterwerte hängt von der Anwendungsdomäne ab und kann bei Bedarf eingegrenzt werden, um den Suchprozess zu beschleunigen.

Die Qualität der entdeckten Prozessmodelle ist stark von den Einstellungen verschiedener Parameter (Hyperparameter) abhängig. Diese Parameter haben Einfluss auf die Datenvorverarbeitung und auch auf die verwendeten Process Mining-Algorithmen. Um die bestmögliche Qualität der Prozessmodelle zu gewährleisten,

wird in diesem Framework eine automatische Hyperparameteroptimierung verwendet. Diese Optimierung geschieht nicht durch ein Abarbeiten aller möglichen Parameterkombinationen, sondern durch eine gezieltere, iterative Suche nach Parameterwerten, welche den zuvor definierten Zielfunktionswert maximieren.

Im vorgestellten Framework kann als Zielfunktionswert *Fitness*, *Precision* oder der F_1-Score genutzt werden.

Für die Durchführung der Hyperparametersuche wird im Framework der Ansatz von Bergstra et al. (2013) verwendet, welcher auf einer bayesianischen Optimierungsmethode basiert. Konkret kommt hier der Algorithmus *Tree of Parzen Estimators (TPE)* zum Einsatz. Die TPE-Methode lernt während der Optimierung schrittweise aus bereits evaluierten Parameterkombinationen und schlägt anschließend neue Parameterkombinationen vor, die mit höherer Wahrscheinlichkeit zu besseren Ergebnissen führen. Unnötige bzw. ungünstige Parameterkombinationen sollen so nach Möglichkeit weniger häufig verwendet werden, was eine Steigerung der Effizienz der Optimierung bedeutet.

Die grundlegende Vorgehensweise der Hyperparameteroptimierung ist in Form einer Feedbackschleife konzipiert, welche in Abbildung 3.1 im oberen Teil des dargestellten Frameworks schematisch dargestellt ist. Im Rahmen dieser Feedbackschleife wird nach jeder Iteration das Qualitätsmaß der erzeugten Prozessmodelle errechnet und zurückgegeben. Basierend auf diesen Ergebnissen entscheidet der TPE-Algorithmus, welche Kombination von Parameterwerten in der nachfolgenden Iteration verwendet werden soll.

Für die Hyperparameteroptimierung ist die Vorauswahl der Parameter, die in den Suchprozess einbezogen werden sollen, eine wichtige Komponente. Diese Vorauswahl sollte durch einen Domänenexperten erfolgen, der einschätzen kann, welche Wertebereiche der einzelnen Parameter für die betrachtete Domäne sinnvoll erscheinen.

Die konkrete Definition dieser Wertebereiche und die Vorauswahl der relevanten Parameter ermöglichen es, den Ansatz flexibel und anwendungsspezifisch zu gestalten. Dadurch ist das hier vorgestellte Verfahren gut auf verschiedene Anwendungsdomänen übertragbar, beispielsweise im Bereich des Smart Home-Umfelds oder anderen Feldern, in denen Sensordaten zur Prozessentdeckung genutzt werden. Für jedes neue Anwendungsszenario können so individuell und effizient die optimalen Parameterkombinationen ermittelt werden.

4 Evaluation

Die Evaluation des Frameworks erfolgt an den in diesem Kapitel vorgestellten Datensätzen. Diese werden zu Beginn der Unterkapitel kurz vorgestellt. Bei der Beurteilung der resultierenden Prozessmodelle ist es wichtig zu beachten, dass sich die Prozessmodelle von klassischen Geschäftsprozessen unterscheiden. Geschäftsprozesse weisen üblicherweise strukturierte Anteile auf, das heißt, sie beschreiben sich wiederholende Ausführungsregeln, die nach einer vordefinierten Struktur ausgeführt werden. Aktivitätsabläufe, insbesondere im privaten Smart Home-Kontext, folgen keinen bestimmten Regeln oder Mustern, wie es bei Geschäftsprozessen üblicherweise der Fall ist.

Die vorgestellte Methode zur Erkennung von Prozessmodellen ist nicht auf die hier vorgestellten Datensätze beschränkt. Grundsätzlich lässt sich das Framework auf jeglichen Sensorereignislogs anwenden. Voraussetzung für die erfolgreiche Anwendung des Frameworks ist, dass die Datensätze die folgenden grundlegenden Eigenschaften wie Zeitstempel, Sensor-ID und Sensorstatus enthalten. Zusätzlich zu diesen Eigenschaften für das Ereignislog müssen die Datensätze Informationen über die Lage der Sensoren enthalten. Bei getragenen Sensoren müssen zusätzliche Informationen vorliegen, an welchen Körperstellen die entsprechenden Sensoren angebracht sind. Bei stationären Sensoren muss es analog dazu einen Lageplan geben, in dem alle Sensoren eingezeichnet sind, oder zumindest die relative Position der Sensoren zueinander.

Aufgrund der zahlreichen Möglichkeiten, wie sich ein Mensch durch den Raum bewegen kann, werden die resultierenden Prozessmodelle immer unstrukturierter sein als Geschäftsprozessmodelle. Es ist also zu erwarten, dass die entdeckten Prozessmodelle eine unübersichtliche Struktur aufweisen.

Die Anwendung von Process Mining auf Sensor-Ereignisdaten zeigt, dass es keine allgemeingültige Parameterkombination für das Erkennen von Prozessen aus unstrukturierten Daten gibt: Die Wahl der Parameter hängt von der Datengrundlage

ab. Die flexible und variable Parameterwahl führt zu besseren Ergebnissen als eine vordefinierte Parameterkombination.

Sensoren im CASAS-Datensatz sind stationär im Smart Home angebracht. Jeder Raum ist mit mehreren Bewegungssensoren an der Decke ausgestattet. Da es mehrere Sensoren pro Raum gibt, ist die Ortung einer Person nicht auf den Raum bestimmt, sondern es ist möglich, den Standort einer Person innerhalb eines Raumes zu bestimmen. Durch die Deckenmontage müssen die Bewohner und Bewohnerinnen keine Sensoren am Körper tragen. Bei dieser Art von Überwachung ist generell von einer höheren Akzeptanz auszugehen (Choukou et al. 2021). Durch die Stationarität der Sensoren ist es nicht notwendig, dass die beobachteten Personen daran denken müssen, Sensoren regelmäßig zu tragen, wie dies bei getragenen Sensoren der Fall wäre. Durch die lückenlose Aufzeichnung der stationären Sensoren im Vergleich zu getragenen Sensoren ist damit zu rechnen, dass die Aufzeichnungen vollständiger sind.

Ziel der Anwendung des Frameworks auf die CASAS-Datensätze ist die Identifikation wiederkehrender Routinen in Smart-Home-Umgebungen.

Aktivitäten wie *essen*, *sitzen* und *duschen* können aus den Sensorereignisdaten abgeleitet werden. Diese Aktivitäten werden dann zu Aktivitätssequenzen angeordnet, die wiederum zu täglichen Routinen zusammengefasst werden. Die Daten bestehen aus Zeitstempeln, den Sensor-IDs und dem aktuellen Status des Sensors (aktiviert = 1; nicht aktiviert = 0). Tabelle 4.1 zeigt einen Auszug des Sensorereignisprotokolls des CASAS-Datensatzes. Abbildungen 4.2 & 4.1 zeigen den Grundriss der Testumgebungen *Kyoto* und *Aruba* inklusive der installierten Sensoren.

Aruba Dieser Datensatz wurde in der Aruba-Testumgebung aufgezeichnet (Abbildung 4.1). Im Vergleich zur Kyoto-Testumgebung sind in dieser Umgebung die Sensoren in deutlich größerem Abstand zueinander angebracht. Die Überwachungsdichte ist also insgesamt deutlich niedriger. Dies erschwert eine genauere Analyse aus zweierlei Gründen:

Präzise Ortung der Entitäten: Es kann oft nur festgestellt werden, dass eine Entität sich in einem Raum befindet, der genaue Standort im Raum ist jedoch schwer feststellbar.

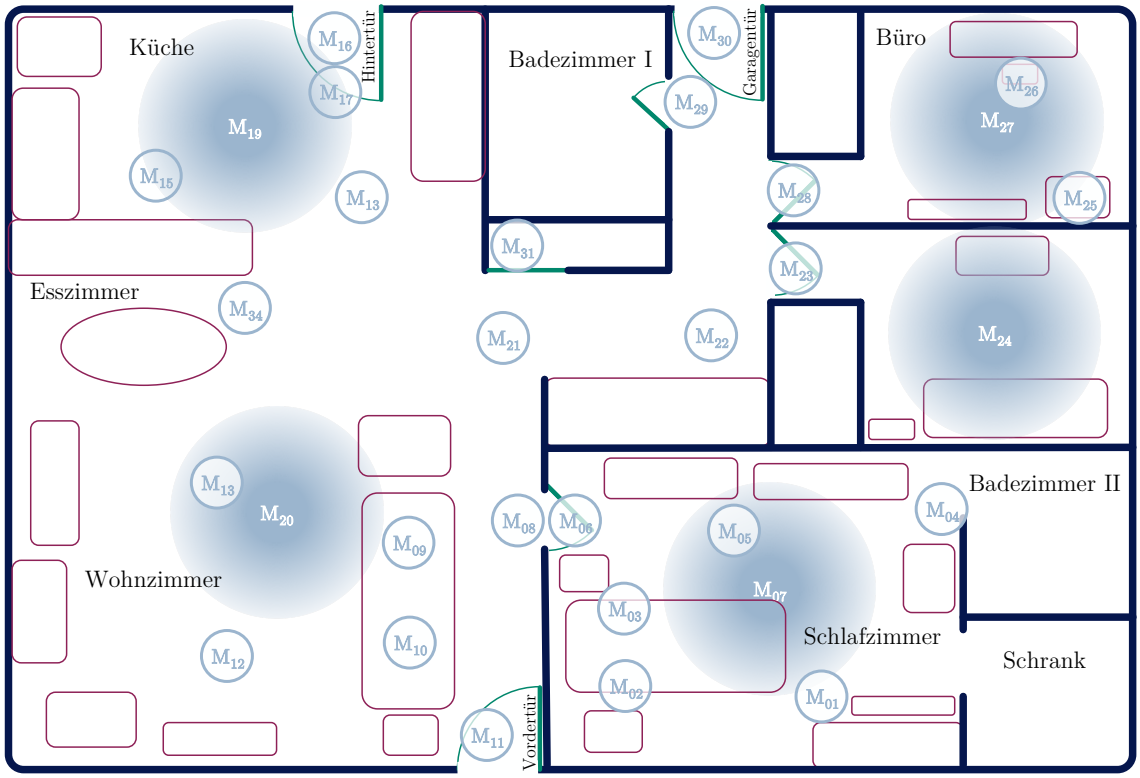


Abbildung 4.1: Testumgebung `Aruba`

Unterscheidung der Entitäten: Durch die grobe Ortung der Entitäten im Raum ist eine Unterscheidung der Entitäten untereinander auch nur schwer möglich.

Ausgewertet wurde der folgende Datensatz aus der Aruba-Testumgebung:

Aruba-19: Der Datensatz Aruba-19 ist im Erfassungszeitraum nur von einer Person bewohnt worden. Eine Unterscheidung zwischen Entitäten entfällt.

Kyoto Abbildung 4.2 zeigt das Sensorlayout der Testumgebung Kyoto. Es gibt in der Testumgebung zwei Stockwerke. Im Erdgeschoss befindet sich eine Küche, ein Wohn- und Esszimmer und eine Abstellkammer. Im Obergeschoss befinden sich zwei Schlafzimmer, ein Badezimmer und ein unüberwachter Raum. Die Testumgebung zeichnet sich dadurch aus, dass jeder Raum mit mehreren Bewegungsmeldern ausgestattet ist, so dass nicht nur festgestellt werden kann, ob sich eine Person im entsprechenden Raum aufhält, sondern wo im Raum sich eine Person aufhält. Beide Datensätze enthalten jeweils mehr als 150 000 Sensorereignisse.

Tabelle 4.1: Auszug aus dem Ereignisprotokoll CASAS-*Kyoto-05* eines Standortsensors, das als Eingabe für das Framework verwendet wird.

Zeitstempel	SensorID	Status
...	...	...
2008-03-20 16:23:04.679667	M010	1
2008-03-20 16:23:08.552586	M010	0
2008-03-20 16:23:27.943178	M010	1
2008-03-20 16:23:30.578482	M010	0
2008-03-20 16:23:37.688497	M010	1
2008-03-20 16:23:39.545067	M010	0
2008-03-20 16:23:39.732019	M010	1
2008-03-20 16:23:40.021204	M006	1
2008-03-20 16:23:44.499433	M009	1
2008-03-20 16:23:44.080557	M008	1
2008-03-20 16:23:44.080557	M010	0
2008-03-20 16:23:45.698575	M015	1
2008-03-20 16:23:46.019598	M016	1
2008-03-20 16:23:46.545595	M006	0
2008-03-20 16:23:46.957546	M017	1
2008-03-20 16:23:47.368475	M008	0
2008-03-20 16:23:47.825345	M009	0
...	...	...

Kyoto-05: Während des Erfassungszeitraumes lebte nur eine Person in dieser Testumgebung. Allerdings gab es sporadische Besuche in der Testumgebung, so dass sich gelegentlich auch mehrere Personen im Raum aufhielten. Durch die nahe beieinander angeordneten Bewegungsmelder können in dieser Testumgebung relativ genaue Rückschlüsse anhand der Sensordaten gezogen werden, wo sich eine Person im Raum aufhält. So können Zeiträume, in denen mehrere Personen anwesend waren, herausgefiltert werden.

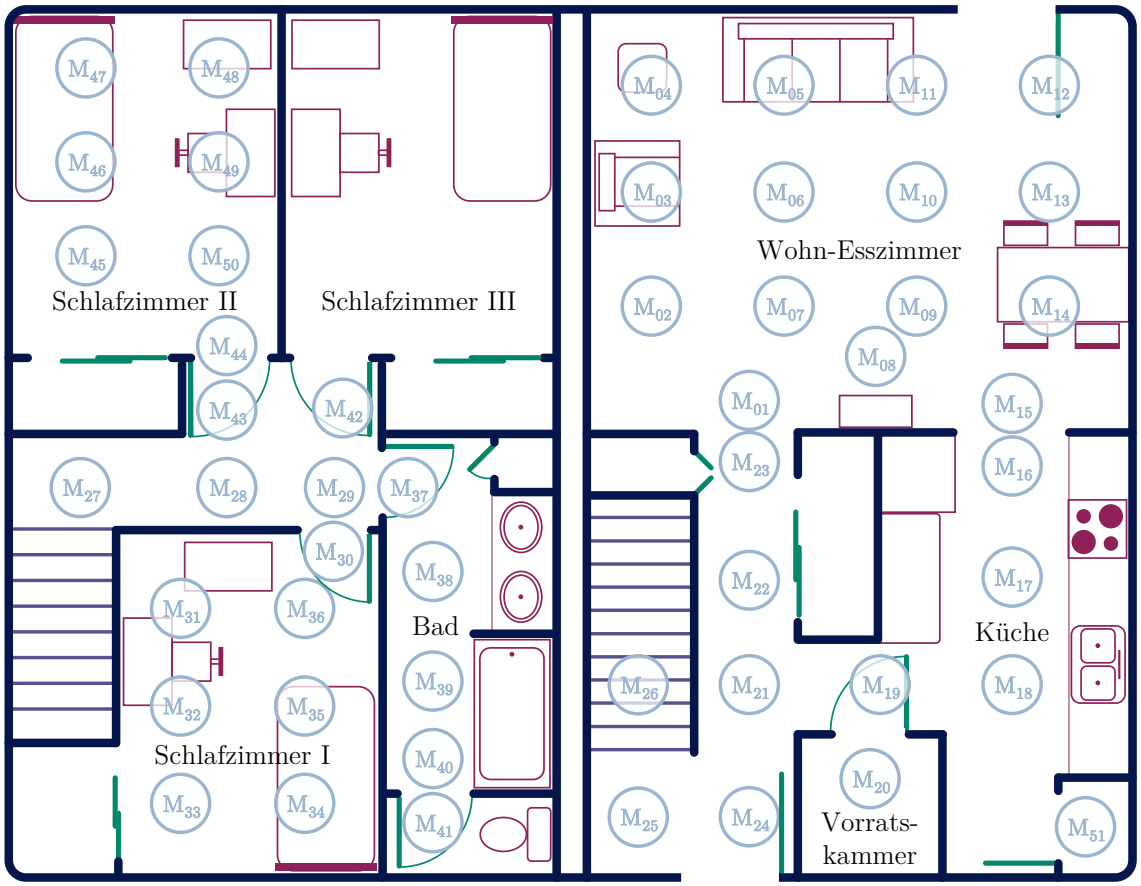


Abbildung 4.2: Testumgebung `Kyoto`

Kyoto-20: Der Unterschied zum Datensatz Kyoto-05 ist, dass während des Erfassungszeitraumes zwei Personen gleichzeitig im Haus lebten. Da die beiden Personen keinerlei Unterscheidungsmerkmale in den aufgezeichneten Daten aufweisen, ist es anspruchsvoll, die Sensoraktivierungen den jeweiligen Personen zuzuordnen.

4.1 Eindeutige Identifikation von Entitäten

Für die Anwendung des Frameworks bei eindeutiger Entitätsidentifizierung wurden zwei Datensätze (**Kyoto-05** und **Aruba-19**) ausgewählt, die vom CASAS Smart Home Projekt Cook and Schmitter-Edgecombe (2009) zur freien Verfügung gestellt wurden. Da es sich bei diesem Evaluationsdurchlauf um Aktivitätserkennung mit eindeutiger Entitätsidentifizierung handelt (im Gegensatz zu Kapitel 4.2), wurden Zeiträume von der Auswertung ausgeschlossen, in denen die Sensoraktivierungen

nicht eindeutig einer einzelnen Person und deren Aktivität zugeordnet werden konnten. Dies trifft zu, wenn die Person, die das Smart Home während des Zeitraumes des Experiments bewohnt, Besuch empfängt, oder Personen im Haus anwesend sind, die das Experiment überwachen. Um das vorgestellte Framework zu validieren, wurde eine Hyperparametersuche mit 200 Iterationen durchgeführt. Zur Einordnung der Ergebnisse wurden weitere 200 Iterationen mit Zufallszuordnungen durchgeführt, die im Folgenden als *Baseline* bezeichnet werden.

Durch die Nutzung der vektorisierten Operationen in NumPy (Harris et al. 2020) und Pandas (McKinney et al. 2010) konnte die Berechnung jeder Iteration effizient durchgeführt werden, unabhängig von der jeweiligen Parameterkonfiguration und Datensatzgröße. Aufgrund begrenzter Arbeitsspeicherkapazität musste der Aruba-19-Datensatz auf 1 250 000 Ereignisse begrenzt werden.

Um die Auswahl der Hyperparameter zu bewerten, wurde der Einfluss einzelner Parameter auf den F_1-Wert untersucht. Hierbei wurde als Optimierungskriterium der F_1-Wert für die Prozessmodelle der Tagesroutinen gewählt.

In Tabelle 4.2 sind die Parameter aufgelistet, die für die Hyperparametersuche genutzt wurden. Zusätzlich sind die Auftrittshäufigkeiten für die einzelnen Durchläufe angegeben. Hervorzuheben ist in der Tabelle 4.2 der Zahlenwert des Heuristic Miners (Spalte 1, Zeile 7). Dieser wurde im Hauptdurchlauf über viermal so oft von der Hyperparametersuche gewählt, wie der Inductive Miner (163 zu 37). Zudem wurden die Tage deutlich häufiger als zusammenhängende Einheit behandelt (Day Partition = 1), anstatt sie in mehrere Routinen zu unterteilen.

Insgesamt wurden für den Kyoto-05-Datensatz drei Auswertungsdurchläufe verwendet:

- Hauptdurchlauf
- Heuristic Miner Optimierung
- Baseline

Die Ergebnisse der Evaluation werden im Folgenden quantitativ anhand der errechneten F_1-Werte und qualitativ anhand der Plausibilität der Prozessmodelle beurteilt.

Tabelle 4.2: Hyperparameter und deren Auftrittshäufigkeit für Kyoto-05

		Haupt-durchlauf	Haupt-durchlauf HM	HM Opti-mierung
Part.	**Aktivierungen**	29	71	119
	Zeit	83	23	30
	Zufall	88	69	51
Vect.	**Quantity**	83	70	31
	Time	88	21	40
	Q. & T.	29	72	129
Miner	**Heuristic**	163	163	200
	Inductive	37	-	-
Clustering	**k-Means**	26	22	25
	k-medoids	26	16	39
	k-mediods dist.	78	69	107
	SOM	70	56	29
Day Partitions	**1**	106	88	97
	2	46	17	25
	3	24	20	31
	4	23	16	24
	5	24	22	23
Week Sep.	**none**	33	25	118
	weekday	134	112	50
	workday	33	26	32
Multi Room	**True**	38	13	69
	False	162	150	131

Kyoto-05: Hauptdurchlauf

Bei dem Hauptdurchlauf werden alle zur Verfügung stehenden Parameter (siehe Tabelle 4.2) verwendet.

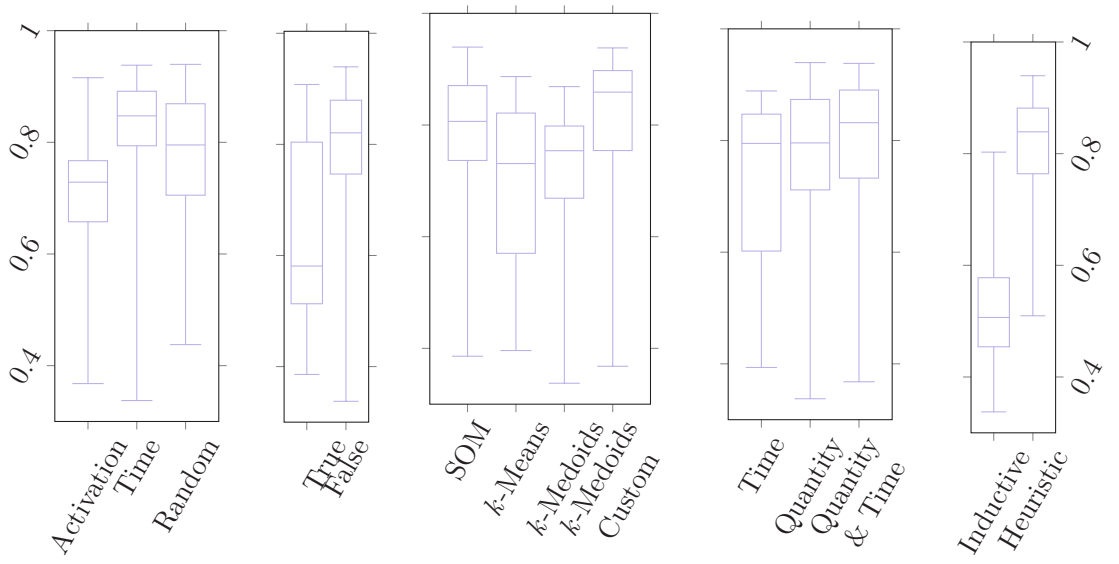
Abbildung 4.3 zeigt fünf verschiedene Boxplots, die die einzelnen Parameter beschreiben. Diese zeigen signifikante Unterschiede zwischen den untersuchten Methoden und Parametereinstellungen. Bei Auswahl der Trace Partition Methode werden mit dem Parameter *Time* und dem Median von 0.8474 die besten Ergebnisse erzielt. Bei dem Parameter, der bestimmt, ob Aktivitäten über mehrere Räume zulässig sind, zeigt sich anhand der Boxplots, dass ein deutlich höherer F_1-Wert erzielt werden kann (Median von 0.8208), wenn Aktivitäten auf einen Raum beschränkt sind. Bei den Clustering-Methoden wird mit dem k-Medoids Verfahren in Kombination mit eigener Distanzfunktion der höchste Median mit 0.8589 erreicht. Für die Vektorisierungsverfahren liefert die Kombination aus *Quantity & Time* die besten F_1-Werte (Median: 0.8319). Bei den Miner-Typen ist der Heuristic Miner (Median: 0.8391) dem Inductive Miner deutlich überlegen. Die Ergebnisse aus Abbildung 4.3e stellen die Eignung des Inductive Miners für den vorliegenden Datensatz in Frage. Der beste F_1-Wert, der mithilfe des Inductive Miners erreicht wird, ist mit 0.8027 unter dem Medianwert des Heuristic Miners (0.8391).

Basierend auf dieser Erkenntnis erfolgt die weitere Auswertung in den folgenden beiden Unterabschnitten:

Kyoto-05: Hauptdurchlauf gefiltert nach Heuristic Miner Dieser Unterabschnitt enthält die gleichen Ergebnisse wie der vorherige Hauptdurchlauf, jedoch ohne jene Iterationen, in denen der Inductive Miner ausgewählt wurde.

Kyoto-05: Heuristic Miner Optimierung In diesem Unterabschnitt wird ein vollständig neuer Durchlauf vorgestellt, bei dem der Heuristic Miner explizit nicht als möglicher Parameter vorgesehen war.

Eine isolierte Betrachtung ohne den Inductive Miner erscheint sinnvoll, da dessen signifikant schlechtere F_1-Werte die Bewertung der übrigen Parameterkonfigurationen verzerren würden.

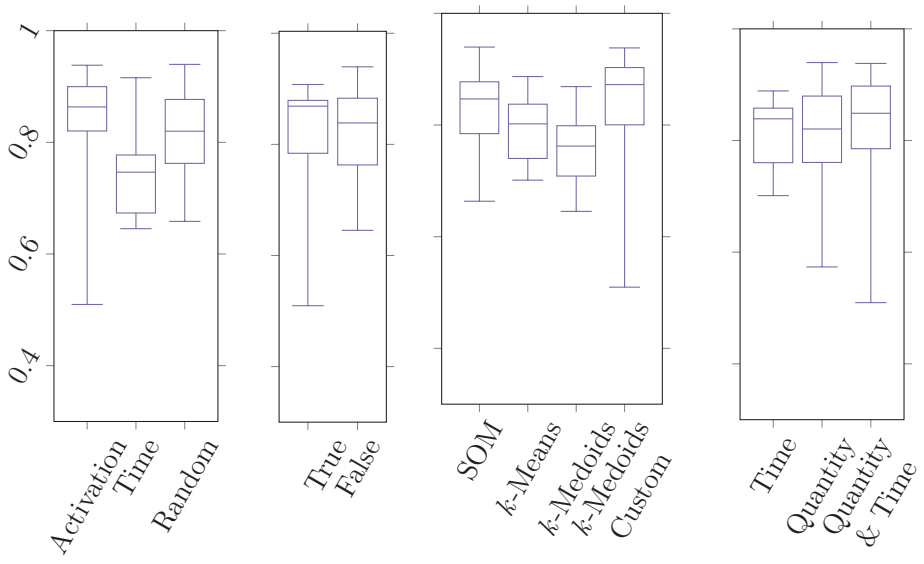


(a) Trace Partition **(b)** Multi-Room **(c)** Clustering-Methode **(d)** Vektorisierungstyp **(e)** Miner Type

Abbildung 4.3: Kyoto-05 Hauptdurchlauf

Kyoto-05: Hauptdurchlauf gefiltert nach Heuristic Miner

Abbildung 4.4 enthält Boxplots der Ergebnisse des Hauptdurchlaufs, bereinigt um alle Iterationen, bei denen der Inductive Miner verwendet wurde. Man erkennt nun, dass der F_1-Wert in allen Parametergruppen für alle Parameter deutlich besser ist als in Abbildung 4.3. Bei der Trace-Partitionierung liefert der Parameter *Activation* mit einem Median von 0.8634 die besten Ergebnisse. Die oberen und unteren Quartile sind denen der anderen beiden Partitionierungsvarianten *Activiation* und *Random* ebenfalls überlegen. Bei dem Parameter, der Aktivitäten in mehreren Räumen zulässt bzw. ausschließt, weisen beide Parameter hohe Medianwerte auf (0.8688 und 0.8386). Bei den Clustering-Methoden wird mit dem k-Medoids-Clustering mit eigener Distanzfunktion der höchste Median von 0.8724 erreicht. Der Interquartilsabstand 0.0973 weist auf eine geringe Streuung der Ergebnisse und damit auf konsistent gute Ergebnisse hin. Für die Vektorisierungstypen werden mit der Kombination *Quantity & Time* mit einem Median von 0.8488 und einem geringen Interquartilsabstand die besten Ergebnisse erzielt.



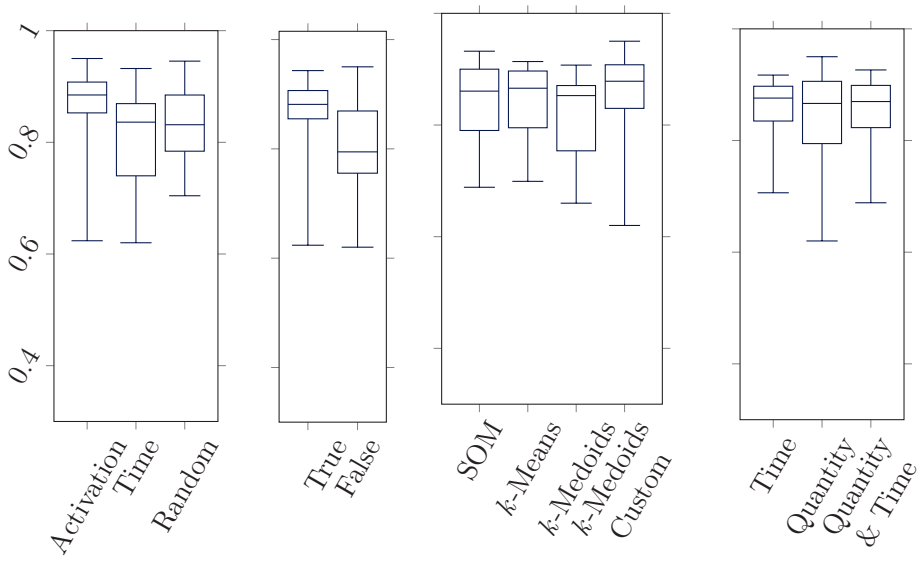
(a) Trace Partition **(b)** Multi-Room **(c)** Clustering-Methode **(d)** Vektorisierungstyp

Abbildung 4.4: Kyoto-05 Hauptdurchlauf gefiltert nach Heuristic Miner

Kyoto-05: Heuristic Miner Optimierung

Schließlich wurde ein weiterer Durchlauf durchgeführt, der den Inductive Miner von vornherein als Parameter ausschloss. Hierdurch wurde die Hyperparametersuche nicht durch die unterdurchschnittlichen Ergebnisse des Inductive Miners (in Bezug auf den F_1-Wert) beeinflusst.

Die Boxplots in Abbildung 4.5 zeigen die Ergebnisse bei der ausschließlichen Verwendung des Heuristic Miners. Bei der Trace Partition erzielt der Parameter *Activations* den höchsten Median (0.8848) und eine sehr geringe Streuung (Interquartilsabstand = 0.0552). Diese deutet darauf hin, dass dieser Parameter zuverlässig gute Ergebnisse liefert. Die Parameter *Activation* und *Random* weisen hingegen ähnliche Mediane auf. Die Streuung des F_1-Werts unter Verwendung dieser beiden Parameter zeigt allerdings eine größere Streuung. In der Multi-Room-Kategorie erreicht True einen leicht höheren Median (0.8814) als False (0.7945), wobei True eine engere Verteilung aufweist.



(a) Trace Partition **(b)** Multi-Room **(c)** Clustering-Methode **(d)** Vektorisierungstyp

Abbildung 4.5: Kyoto-05 Nur Heuristic Miner Durchlauf

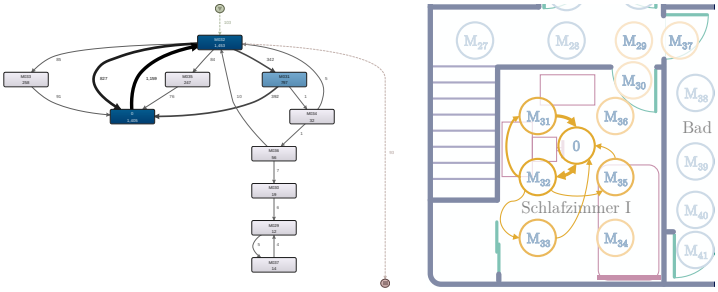
Die Ergebnisse des besten Durchlaufs mit einem F_1-Wert von 0.9355 sind im Folgenden detailliert beschrieben. Die Bezeichnungen der Aktivitäten der einzelnen Cluster sind in Tabelle 4.3 zusammengefasst.

Die Bezeichnung der Aktivitäten erhält man, indem man die Sensoraktivierungen jedes Clusters einzeln analysiert. Hierfür werden die Directly-Follows-Relationen der einzelnen Cluster und der Lageplan des Smart Homes gemeinsam betrachtet.

Tabelle 4.3: Aktivitätenbezeichnung je Cluster

Cluster	Aktivitätsbezeichnung	$\bar{t}_{\text{arithm}}$ [min]	Fälle	Varianten
0	Schreibtischaktivität	15.6	198	198
1	Aufstehen	5.8	114	113
2	Ins Bad gehen	2.6	154	154
3	Schlafen	4.9	130	129
4	Schlafen	14.1	144	144
5	Schlafen	19.6	150	149
6	Schlafen	13.9	25	25
7	Haus betreten (hinten)	2.5	99	64
8	Tisch decken	1.5	10	10
9	Tisch- & Küchenvorbereitung	2,4	126	123
10	Essen zubereiten	1.5	192	192
11	Umhergehen	1.4	244	243
12	Stockwerk wechseln	0.4	438	388
13	Wohnzimmeraktivität	4.6	248	240
14	Auf Couch sitzen	5.5	254	242
15	Auf Couch sitzen	8.0	260	258
16	Auf Couch sitzen	7.6	367	316

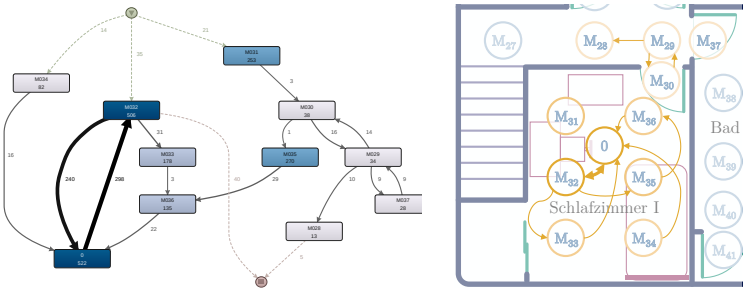
Cluster 0 – Schreibtischaktivität: Die Aktivitäten des Clusters 0 finden zum großen Teil im Schlafzimmer I des Hauses statt. Die aktivierten Sensoren befinden sich hauptsächlich über dem Schreibtisch, was darauf schließen lässt, dass es sich hierbei um *Arbeiten* oder *sonstige Aktivitäten am Schreibtisch* handelt. Die Aktivität ist in Abbildung 4.6 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.6: Heuristic Miner Optimierung – Cluster 0

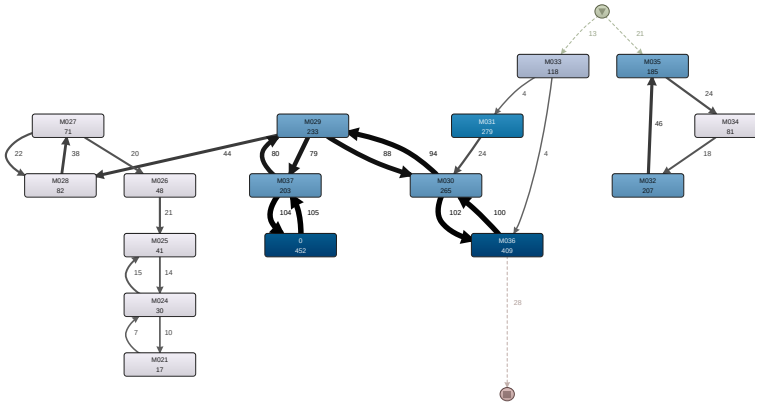
Cluster 1 – Aufstehen: Die Aktivitäten des Clusters 1 finden ebenfalls wie die Aktivitäten des Clusters 0 im Schlafzimmer I des Hauses statt. Die Dauer der Aktivierungen ist mit unter sechs Minuten deutlich geringer als in Cluster 0. Zudem ist hier überwiegend der Sensor `M32` aktiviert, der mittig im Raum hängt. Die Sensoraktivierungen in diesem Cluster finden vorwiegend früh morgens statt. Dieses Cluster kann deshalb mit der Aktivität *Aufstehen* klassifiziert werden.



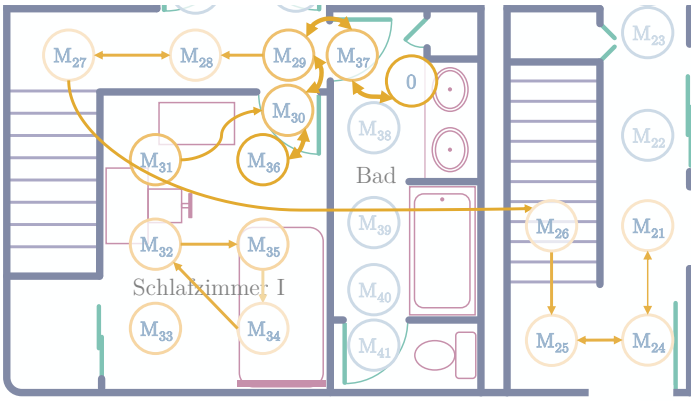
(a) DFG **(b)** Lageplan

Abbildung 4.7: Heuristic Miner Optimierung – Cluster 1

Cluster 2 – Ins Bad gehen: Die Aktivitäten, zusammengefasst in Cluster 2, beziehen sich auf den Bereich zwischen Schlafzimmer I und Bad. Besonders aktiv sind die Sensoren `M036`, `M030`, `M029` und `M037`. Diese Sensoren liegen auf dem Weg vom Schlafzimmer über den Flur ins Badezimmer. Daraus lässt sich schließen, dass es sich bei den aufgezeichneten Bewegungen um die Aktivität *Ins Bad gehen* handelt. Diese Aktivität tritt entweder im Rahmen einer morgendlichen Routine oder bei anderen regelmäßigen Wege ins Bad auf. Die Aktivität ist in Abbildung 4.8 dargestellt.



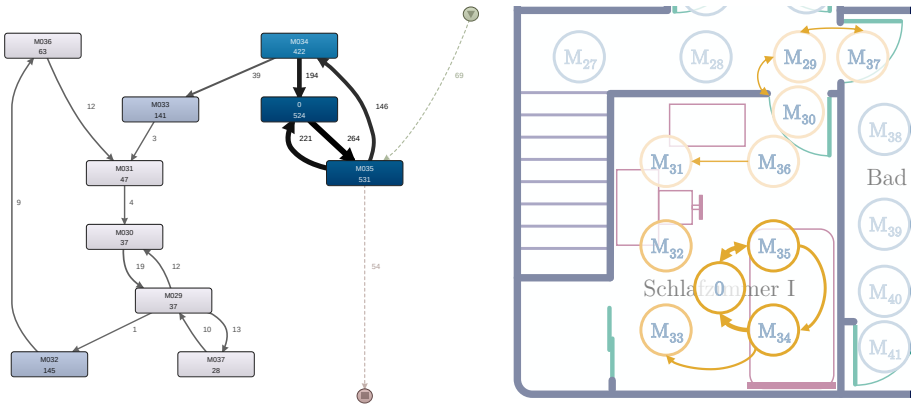
(a) DFG



(b) Lageplan

Abbildung 4.8: Heuristic Miner Optimierung – Cluster 2

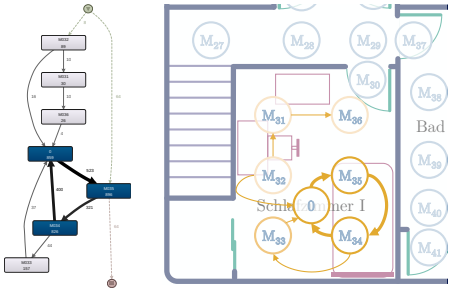
Cluster 3 – Schlafen: Die Aktivitäten, die in Clusters 3 zusammengefasst sind, finden nahezu ausschließlich im Schlafzimmer I statt. Besonders aktiv sind die Sensoren `M033`, `M034` und `M035`. Diese Sensoren befinden sich oberhalb des Betts und in dessen unmittelbarer Nähe. Die Beschränkungen der Aktivität auf diesen Bereich und die geringe Beteiligung anderer Räume deuten darauf hin, dass es sich um eine stationäre Aktivität handelt. Aufgrund des Standorts der beteiligten Sensoren und der geringen Bewegungsreichweite lässt sich die Aktivität am besten als *Schlafen* klassifizieren. Die Aktivität ist in Abbildung 4.9 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.9: Heuristic Miner Optimierung – Cluster 3

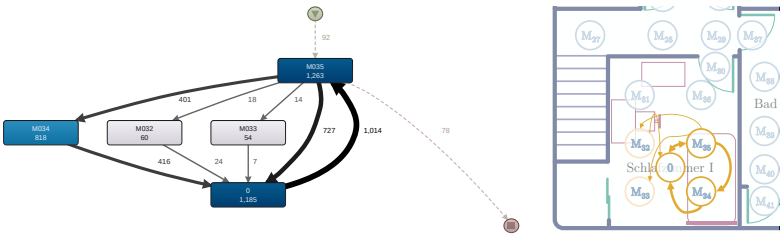
Cluster 4 – Schlafen: Die Aktivitäten des Clusters 4 konzentrieren sich ähnlich wie in Cluster 3 deutlich auf Schlafzimmer I. Die am häufigsten aktivierten Sensoren sind ebenfalls die Sensoren über und neben dem Bett (`M033`, `M034` und `M035`). Die Aktivierung dieser Sensorgruppe und fehlende Aktivitäten in anderen Gebäudeteilen deuten auf eine stationäre Nutzung des Schlafzimmers. Das Verhalten lässt sich daher am wahrscheinlichsten als *Schlafen* klassifizieren. Die Aktivität ist in Abbildung 4.10 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.10: Heuristic Miner Optimierung – Cluster 4

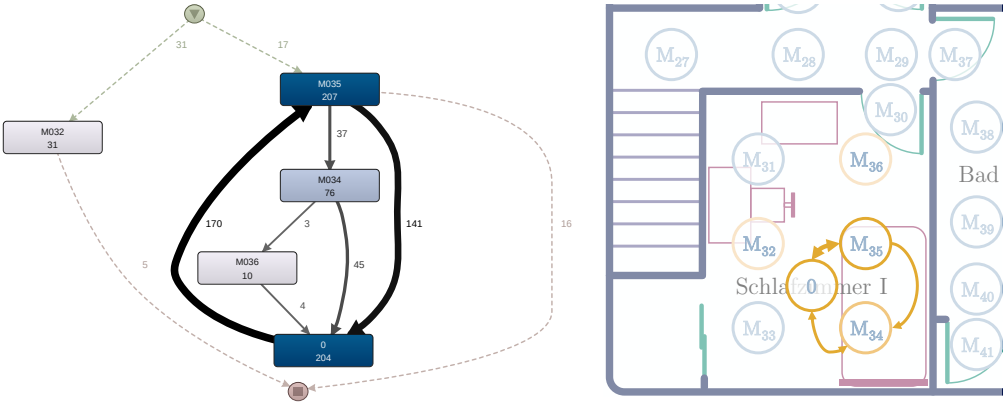
Cluster 5 – Schlafen: Die Aktivitäten des Clusters 5 beschränken sich ähnlich wie die Cluster 3 und 4 vollständig auf das Schlafzimmer I. Besonders häufig aktiviert wurden die Sensoren in Bettnähe (`M033`, `M034` und `M035`). Die Verteilung der Sensoraktivierungen lässt kaum Bewegung außerhalb des Betts vermuten. Damit ergibt sich das eindeutige Aktivitätsmuster, nämlich *Schlafen*. Die Aktivität ist in Abbildung 4.11 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.11: Heuristic Miner Optimierung – Cluster 5

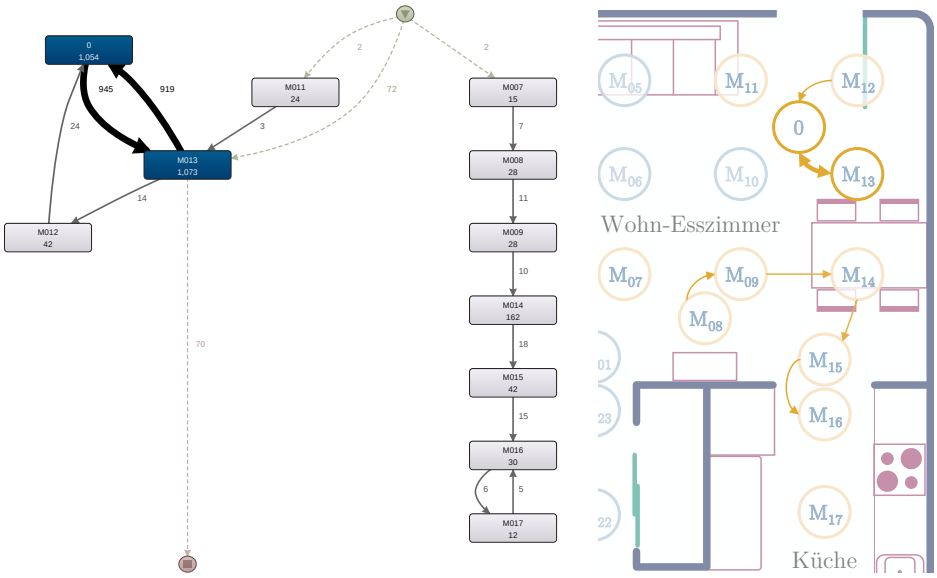
Cluster 6 – Schlafen: Cluster 6 ist das Letzte der insgesamt vier Cluster, die mit der Aktivität *Schlafen* gelabelt werden kann. Die Aktivitäten des Clusters 6 konzentrieren sich ebenfalls ausschließlich auf das Schlafzimmer I. Insbesondere die Sensoren am Bett (`M033`, `M034` und `M035`) sind häufig aktiviert. Die fehlende Beteiligung anderer Räume lässt auf eine überwiegend ruhende Aktivität schließen. Die Aktivität ist in Abbildung 4.12 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.12: Heuristic Miner Optimierung – Cluster 6

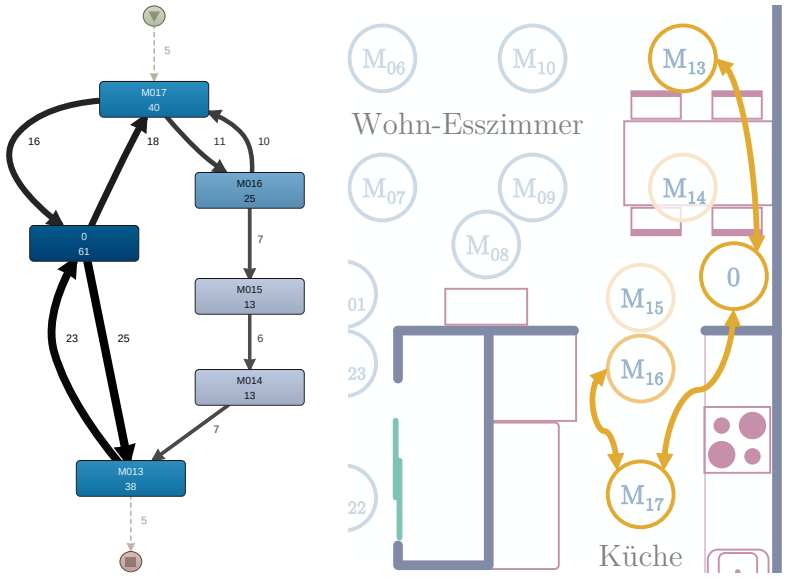
Cluster 7 – Haus betreten (Hintereingang): Die Aktivitäten des Clusters 7 konzentrieren sich auf den hinteren Bereich des Hauses im Wohnzimmer, insbesondere rund um den Sensor `M013`. Dieser befindet sich direkt an der Hintertür, die als Zugang zum Haus genutzt werden kann. Die nachfolgenden Sensoren `M011`, `M016` und `M017` liegen in Richtung des Hausinneren. Die Aktivierungssequenz deutet auf eine typische Bewegung von außen über die Hintertür ins Haus hinein. Es liegt daher nahe, diese Aktivität als *Haus betreten* zu benennen. Die Aktivität ist in Abbildung 4.13 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.13: Heuristic Miner Optimierung – Cluster 7

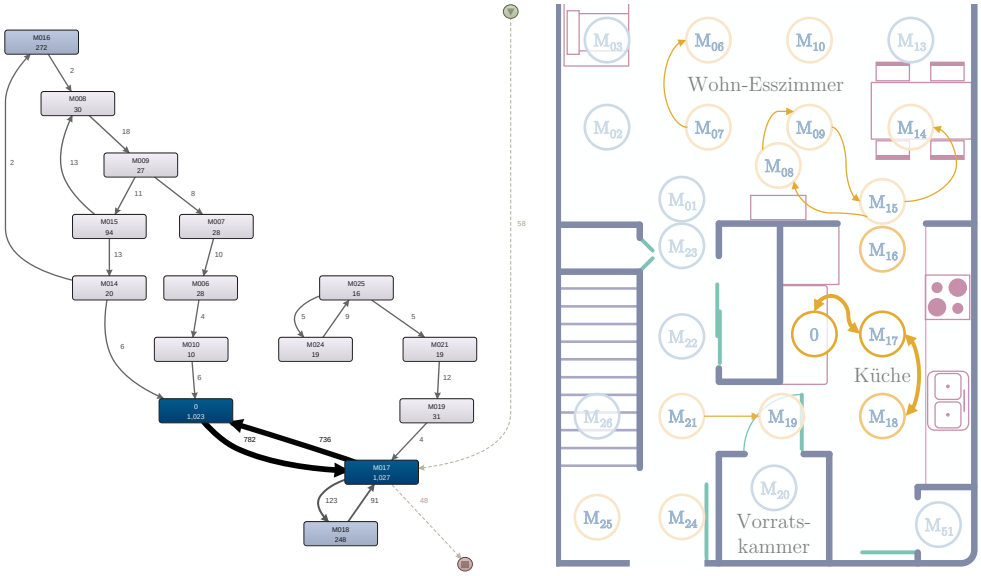
Cluster 8 – Tisch decken: Die Aktivitäten des Clusters 8 sind auf einen Bereich rund um den Esstisch im Wohn-Esszimmer beschränkt. Besonders aktiv sind die Sensoren `M013` (in Tischnähe) und `M016` und `M016` (Küche und Übergang von Küche zu Wohn-Esszimmer). Die Bewegung entlang dieser Sensoren spricht für eine gezielte Aktivität am Esstisch, ohne Bewegungen zwischen Küche und Wohn-Essbereich. Daraus lässt sich schließen, dass dieses Cluster die Aktivität *Tisch decken* beschreibt. Die Aktivität ist in Abbildung 4.14 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.14: Heuristic Miner Optimierung – Cluster 8

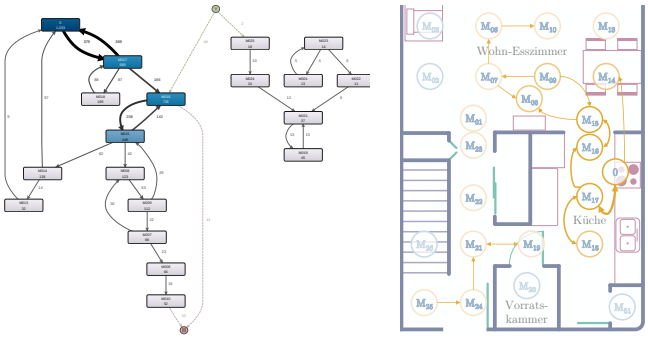
Cluster 9 – Tisch decken und Vorbereitungen in der Küche: Die Aktivitäten des Clusters 9 umfassen sowohl den Küchenbereich als auch den Essbereich im Wohn-Esszimmer. Im Vergleich zu Cluster 8 überwiegen in diesem Cluster jedoch die Aktivitäten in der Küche. Das wiederholte Wechseln zwischen Küche und Tisch deutet auf eine kombinierte Aktivität hin. Die wahrscheinlichste Interpretation ist daher *Tisch decken und Vorbereitungen in der Küche*. Die Aktivität ist in Abbildung 4.15 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.15: Heuristic Miner Optimierung – Cluster 9

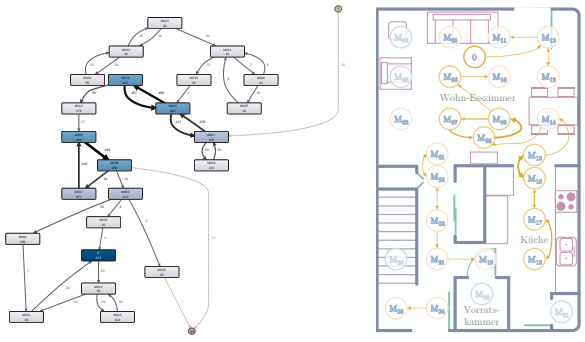
Cluster 10 – Essen zubereiten: Die Aktivitäten des Clusters 10 konzentrieren sich hauptsächlich auf den Küchenbereich. Besonders aktiv sind die Sensoren `M016`, `M017` und `M018`. Die starke Konzentration auf Küche und angrenzende Räume spricht eindeutig für die Aktivität *Essen zubereiten*. Die Aktivität ist in Abbildung 4.16 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.16: Heuristic Miner Optimierung – Cluster 10

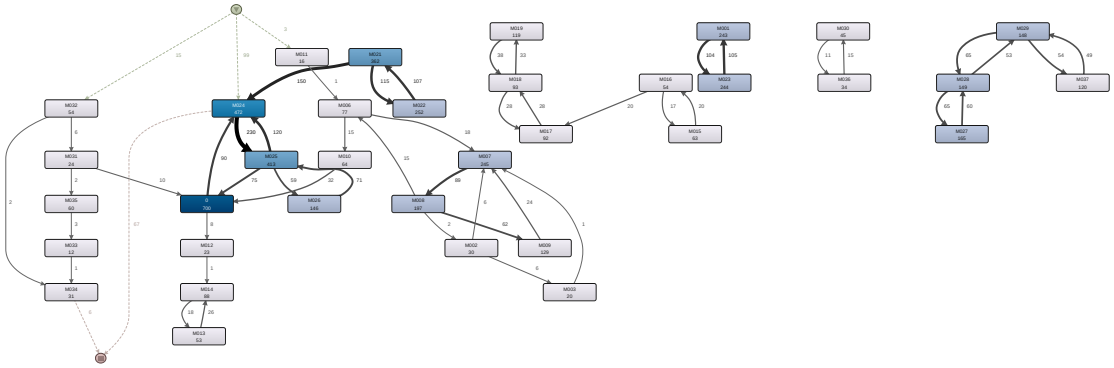
Cluster 11 – Umhergehen: Die Aktivitäten des Clusters 11 sind durch eine Verteilung der Sensoraktivitäten im gesamten Untergeschoss des Hauses charakterisiert. Der DFG zeigt ein weiträumiges, räumlich sehr unbeschränktes Bewegungsmuster. Die Aktivität kann daher als *Umhergehen* interpretiert werden. Die Aktivität ist in Abbildung 4.17 dargestellt.



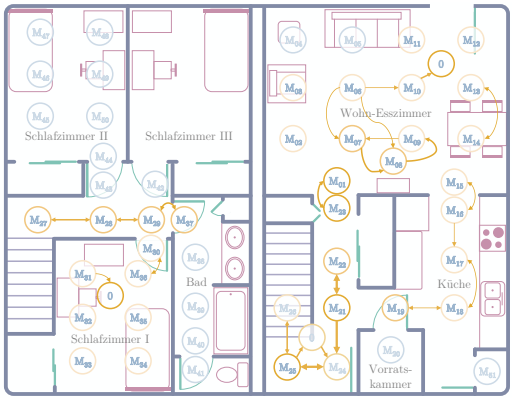
(a) DFG **(b)** Lageplan

Abbildung 4.17: Heuristic Miner Optimierung – Cluster 11

Cluster 12 – Stockwerk wechseln: Die Aktivitäten im Cluster 12 zeigen eine hohe Bewegungsintensität in den Zugangs- und Übergangsbereichen des Hauses, insbesondere bei den Treppensensoren `M025`, `M026` und `M027` sowie in den angrenzenden Fluren.Das spricht dafür, dass es sich um die Bewegung *zwischen zwei Stockwerken* handelt, inklusive des Hinauf- oder Hinabsteigens der Treppe. Die Aktivität ist in Abbildung 4.18 dargestellt.



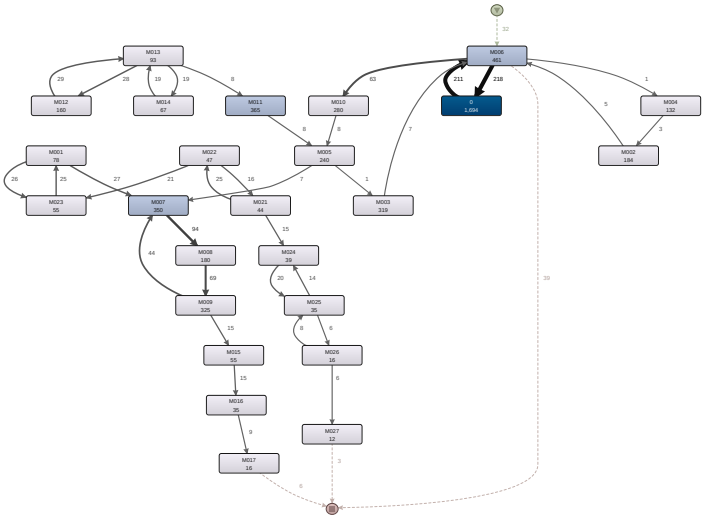
(a) DFG



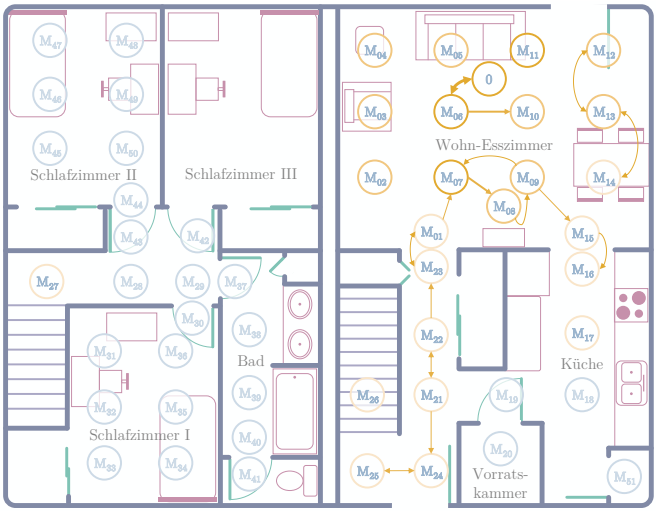
(b) Lageplan

Abbildung 4.18: Heuristic Miner Optimierung – Cluster 12

Cluster 13 – Wohnzimmeraktivität: Die Sensoraktivierungen in Cluster 13 konzentrieren sich stark auf den Wohn-Esszimmerbereich. Besonders aktiv sind die Sensoren, die sich rund um die Sitzmöbel und den zentralen Bereich des Wohnzimmers befinden. Weitere Bewegungen zwischen diesen Positionen deuten auf keine stationäre Aktivität im Wohnzimmer hin. Die Aktivität kann daher eher als eine allgemeine *Wohnzimmeraktivität* interpretiert werden. Die Aktivität ist in Abbildung 4.19 dargestellt.



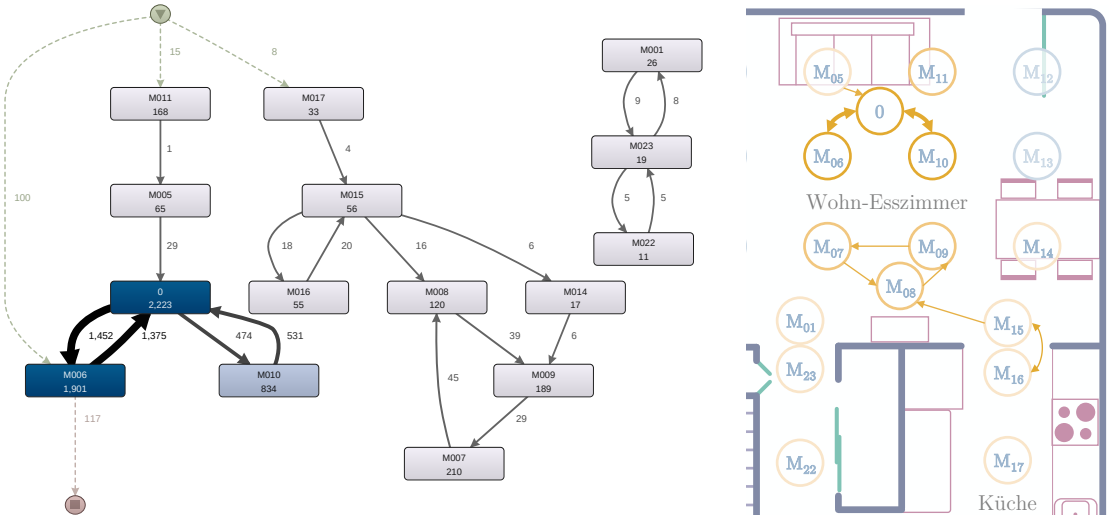
(a) DFG



(b) Lageplan

Abbildung 4.19: Heuristic Miner Optimierung – Cluster 13

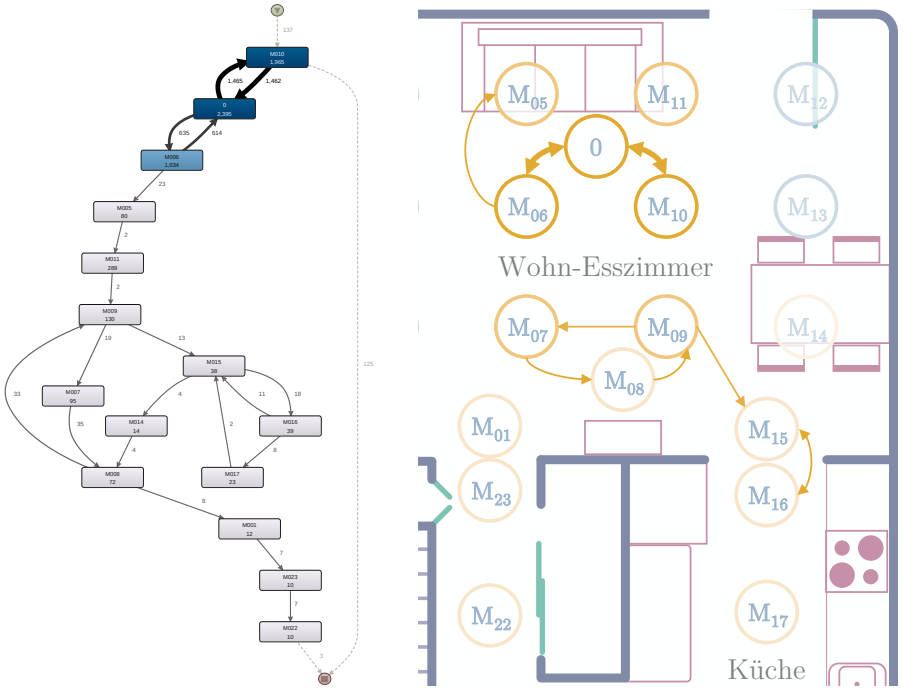
Cluster 14 – Auf der Couch sitzen: Die Aktivitäten des Clusters 14 konzentrieren sich überwiegend auf den Couchbereich des Wohnzimmers, insbesondere im Vergleich zu Cluster 13. Besonders häufig aktiviert sind die Sensoren `M006` und `M010`, die den Bereich über und neben der Sitzfläche der Couch abdecken. Auch benachbarte Sensoren `M005`, `M007` bis `M011` sind beteiligt, welche durch ihre Nähe zum Sitzbereich ein *auf der Couch sitzen* hindeuten könnten. Die Aktivität ist in Abbildung 4.20 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.20: Heuristic Miner Optimierung – Cluster 14

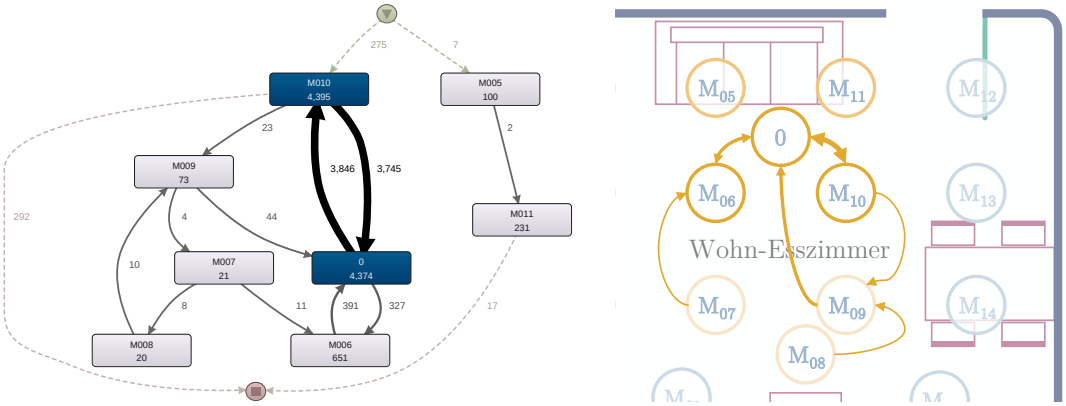
Cluster 15 – Auf der Couch sitzen: Die Sensoraktivierungen in Cluster 15 sind denen aus Cluster 14 sehr ähnlich. Sie weisen eine ausgeprägte Aktivität im Wohnbereich rund um die Couch auf. Die begrenzte räumliche Streuung und die hohe Anzahl von Sensoraktivierungen rund um den Couchbereich deuten auf eine stationäre Aktivität hin. Die passendste Aktivitätsbezeichnung ist *Auf der Couch sitzen*. Die Aktivität ist in Abbildung 4.21 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.21: Heuristic Miner Optimierung – Cluster 15

Cluster 16 – Auf der Couch sitzen: Die Aktivitäten, die in Cluster 16 zusammengefasst sind, sind ähnlich den Aktivitäten in Cluster 14 und 15. Sie beschränken sich auf den Wohnzimmerbereich rund um die Couch. Die Aktivität lässt sich hier ebenfalls als *Auf der Couch sitzen* interpretieren. Die Aktivität ist in Abbildung 4.22 dargestellt.



(a) DFG **(b)** Lageplan

Abbildung 4.22: Heuristic Miner Optimierung – Cluster 16

In Tabelle 4.4 ist die Verteilung der Sensoren auf die einzelnen Cluster dargestellt.

In der obersten Zeile sind die Auftrittshäufigkeiten für den Hilfssensor `M000` pro Cluster dargestellt. Die übrigen Zeilen geben die Häufigkeit der Aktivierungen der jeweiligen Sensoren (`M001`–`M037`) für jedes der Cluster an.

Es lässt sich beobachten, dass die Sensoraktivierungen nicht gleich auf alle Sensoren verteilt sind, sondern einige Sensoren nur in sehr geringer Häufigkeit auftreten. Andere Sensoren wie `M006`, `M010`, `M013`, `M013`, `M017`, `M032`,`M034` und `M035` treten besonders häufig auf.

Es zeigt sich zudem, dass sich die Sensoren jeweils auf bestimmte Cluster konzentrieren, wobei jedes Cluster nur eine begrenzte Auswahl an Sensoren umfasst. Eine auffällige Ausnahme bildet Cluster 12, das alle Sensoren vereint, die in diesem Datensatz von Relevanz sind.

Abbildung 4.23 zeigt den Tagesablauf an einem typischen Montag im Kyoto-05-Datensatz visualisiert als Directly-Follows-Graph. In Abbildung 4.24 ist das dazu-

Sensor	Cluster																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1405	522	452	524	859	1185	204	1054	61	1023	1033	973	700	1694	2223	2395	4374
M001			2					7			13	55	243	78	26	12	
M002								1		2	2	29	30	184	11	1	
M003								1		1		16	20	319	13	1	
M004												1	18	132	6		
M005											3	10	7	240	65	80	100
M006								4		28	64	213	77	461	1901	1034	651
M007								15		28	88	337	245	350	210	95	21
M008			1					28		30	123	408	197	180	120	72	20
M009								28		27	112	486	129	325	189	130	73
M010								1		10	52	138	64	280	834	1965	4395
M011								24	1		12	38	16	365	168	289	231
M012								42	3		10	56	23	160	8	2	
M013								1073	38	3	32	104	53	93	9	5	
M014								162	13	20	138	179	88	67	17	14	1
M015								42	13	94	448	410	63	55	56	38	7
M016								30	25	272	726	483	54	35	55	39	2
M017			1					12	40	1027	689	336	92	16	33	23	1
M018			1					3	14	248	199	120	93	5	6	3	
M019			2						3	31	45	33	119	1	2	1	
M020																	
M021		2	17					1		19	37	43	362	44	8	5	
M022			5					2		2	11	36	252	47	11	10	
M023			2					2		1	14	46	244	55	19	10	
M024		2	30	2				1		19	24	24	472	39	1	3	
M025		3	41	4						16	18	19	413	35	3	1	
M026		4	48	4						1	7	5	146	16		1	
M027		6	71	5						1	7	2	165	12			
M028		13	82	8							2	1	149	5			
M029	12	34	233	37	3						2	1	148	1			
M030	19	38	265	37	7		2						45	1			
M031	797	253	279	47	30	4	3						24				
M032	1453	506	207	145	89	60	31						54				
M033	258	178	118	141	157	54	5						12				
M034	32	82	81	422	826	818	76						31				
M035	247	270	185	531	896	1263	207						60				
M036	56	135	409	63	26	29	10						34				
M037	14	28	203	28	2								120				

Tabelle 4.4: Auftrittshäufigkeit der Sensoren je Cluster für den Durchlauf Heuristic Miner Optimierung. Kyoto-05.

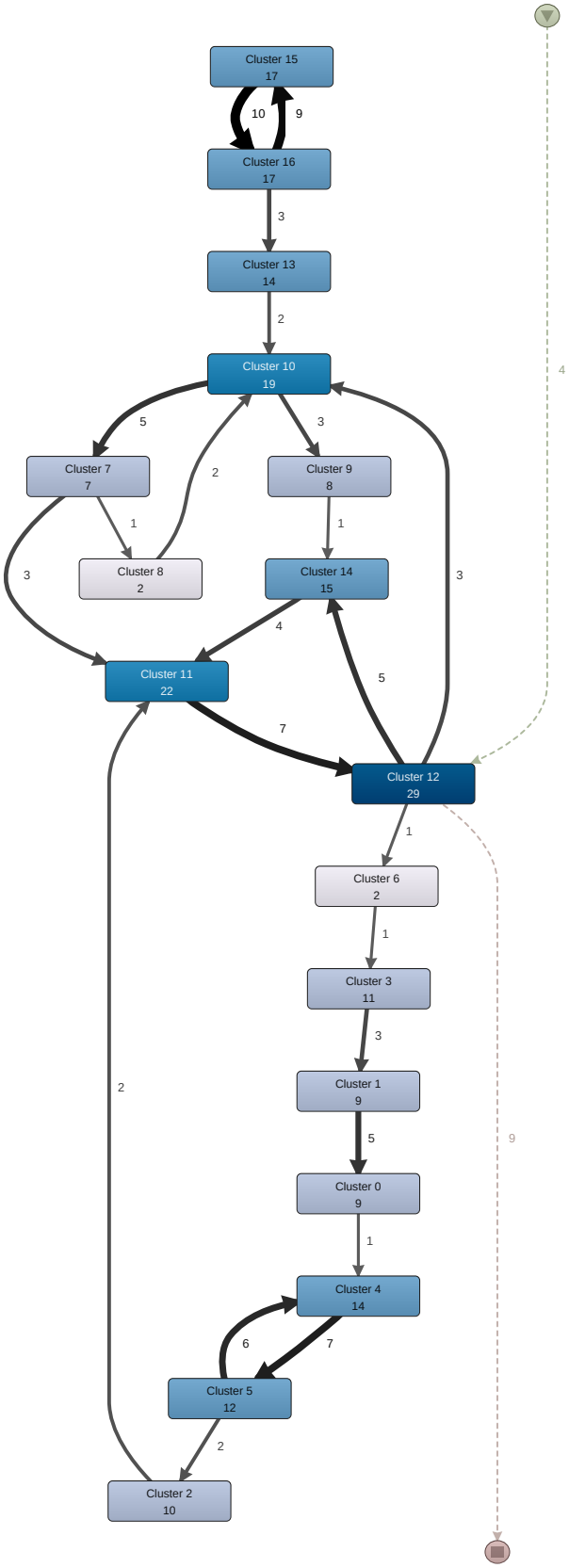


Abbildung 4.23: Tagesablauf für Montag im Kyoto-05 Datensatz (DFG).

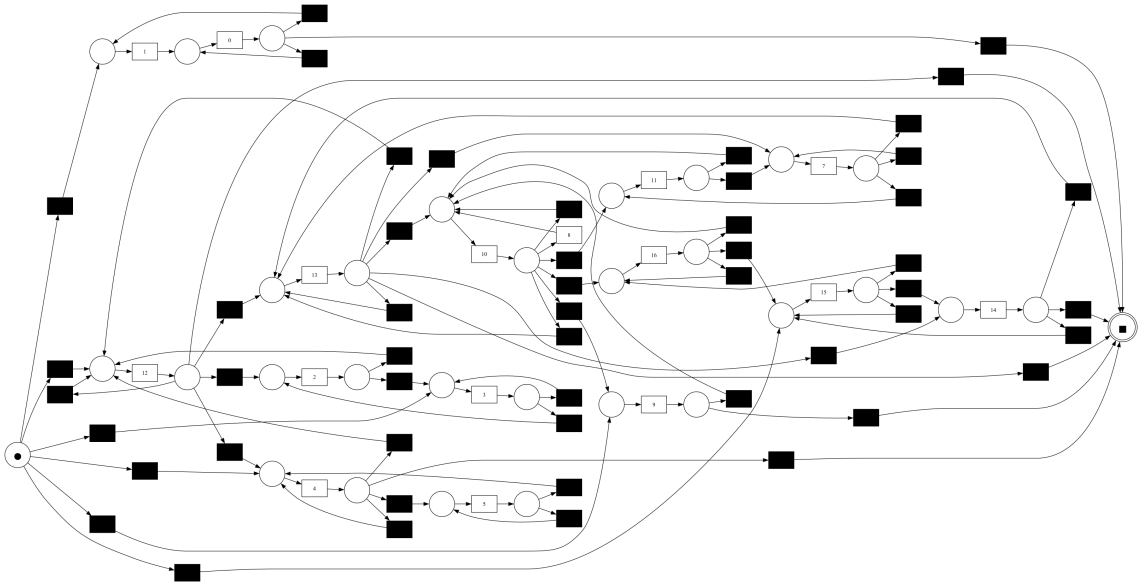


Abbildung 4.24: Tagesablauf für Montag im Kyoto-05 Datensatz (Petri-Netz).

gehörige Petri-Netz gezeigt.

Kyoto-05: Baseline

Für die Baseline-Ausführungsvariante werden die Parameter nach Möglichkeit so gewählt, als gäbe es keine Verarbeitungsschritte im Sensorereignislog. Es findet keine Unterteilung in Routinen statt und die Zuteilung der Fälle zu den Clustern erfolgt nach dem Zufallsprinzip. Dies soll als Vergleich dazu dienen, wie sich die Ergebnisse verbessern, wenn das vorgestellte Framework angewendet wird.

Kyoto-05: Zusammenfassung

Abbildung 4.25 zeigt die Verteilung der F_1-Werte für die vier Ausführungsvarianten der Modellierung auf dem Datensatz Kyoto-05. Die Balkendiagramme veranschaulichen die relative Häufigkeit der erreichten F_1-Werte im Bereich von 0.35 bis 1.

Die Baseline (dunkelrot) konzentriert sich stark auf die Intervalle $[0.8; 0.9)$ und $[0.85; 0.9)$. Niedrigere Scores werden von der Baseline kaum erreicht. Im Vergleich dazu zeigen die Optimierungsläufe (Hauptlauf, Hauptlauf HM, nur HM) eine deutlich breitere Verteilung.

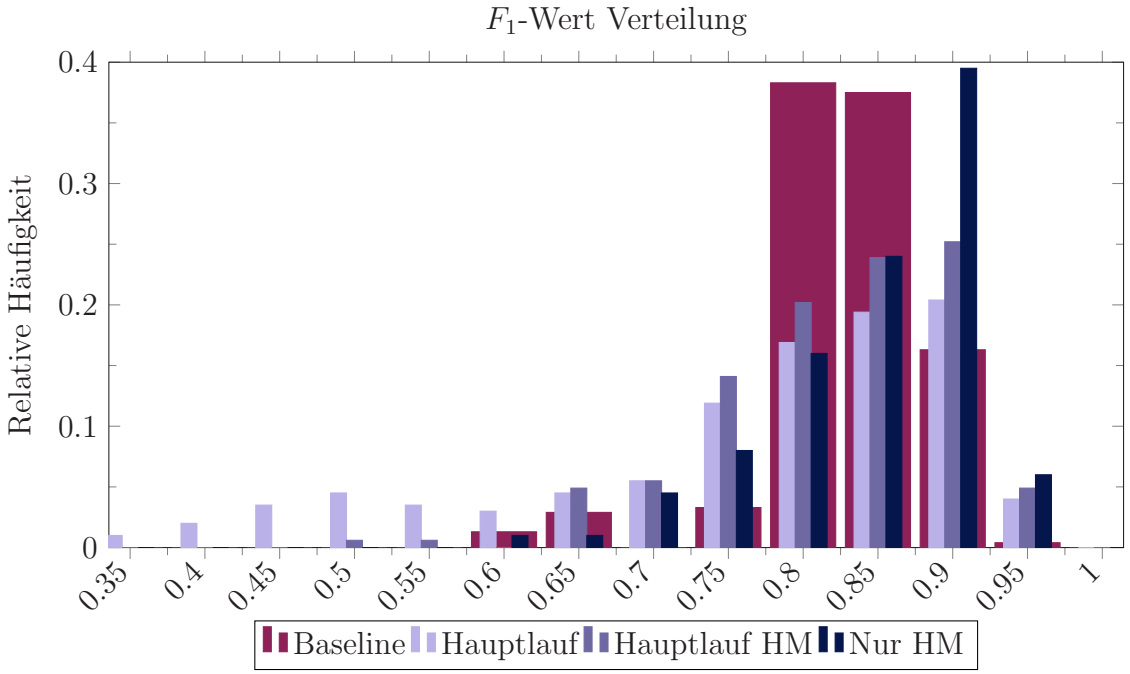


Abbildung 4.25: F_1-Wert für Baseline, Hauptlauf, Hauptlauf (nach HM gefiltert) und Heuristic Miner Optimisation (Kyoto-05).

Der Durchlauf, bei dem ausschließlich der Heuristic Miner (nur HM) verwendet wurde, erzielt den höchsten Anteil an sehr guten Scores im Intervall $[0.9; 0.95)$ (ca. 40%) und übertrifft hier alle anderen Varianten (auch die Baseline).

Tabelle 4.5 zeigt die fünf besten Iterationen. Es scheint, dass eine Partitionierung nach Wochentagen zu konstant besseren Ergebnissen führt. Eine Unterteilung der Tage in Routinen scheint hier jedoch keinen positiven Einfluss auf den F_1-Score der entdeckten Prozessmodelle zu haben.

Aruba-19

Aufgrund der Größe des Datensatzes ist die Verwendung des Inductive Miners aus Effizienzgründen nicht sinnvoll. Der Hauptdurchlauf enthält dementsprechend ausschließlich den Heuristic Miner als Process Discovery-Algorithmus.

Die Anzahl der Cluster oder die Anzahl der Partitionen an einem Tag scheint den resultierenden F_1-Wert nicht stark zu beeinflussen: $corr(F_1, \#\text{Clusters})$ und $corr(F_1, \#\text{Partitions})$ für beide Datensätze im Intervall von $[-0.02; 0.21]$. Die Dauer der Trace oder die Anzahl der Aktivierungen pro Trace beeinflussen den F_1-Score.

Tabelle 4.5: Ergebnisse der besten fünf von 200 Iterationen.

	F_1	Precision	Fitness	Clustering	Trace Length	Number of Clusters	Day Partition	Separator
Main Run	0.9396	0.9352	0.9442	sklearn-SOM	random	20	1	weekday
	0.9383	0.9254	0.9524	sklearn-SOM	random	18	1	weekday
	0.9381	0.9368	0.9407	experimental	30	18	1	weekday
	0.9368	0.8972	0.9806	experimental	5	12	1	weekday
	0.9282	0.922	0.935	experimental	30	18	1	weekday
HM Run	0.9355	0.9359	0.9358	sklearn-SOM		17	1	weekday
	0.9344	0.9113	0.9587	k-Medoids	600	19	1	-
	0.9339	0.9006	0.9715	sklearn-SOM		15	1	weekday
	0.9335	0.9127	0.9553	sklearn-SOM		18	1	-
	0.931	0.9137	0.951	sklearn-SOM		15	1	weekday
Baseline	0.9334	0.9752	0.8949	baseline	baseline	6	1	-
	0.9240	0.9476	0.9015	baseline	baseline	6	1	-
	0.9201	0.9437	0.8977	baseline	baseline	7	1	-
	0.9153	0.9476	0.8851	baseline	baseline	6	1	-
	0.9140	0.8416	1.0000	baseline	baseline	6	1	-

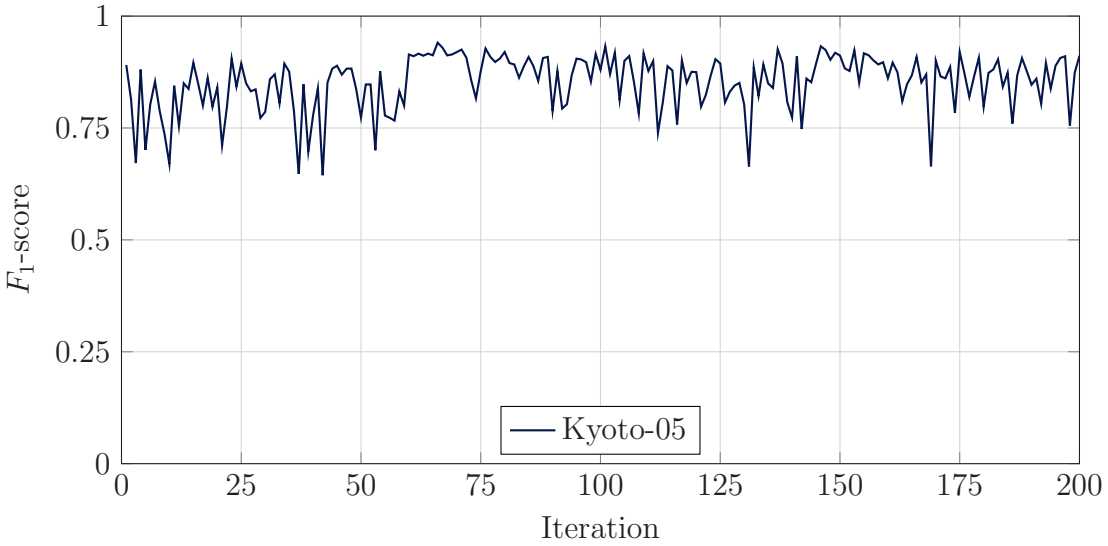


Abbildung 4.26: Funktionswert der Hyperparameter-Optimierung (Kyoto)

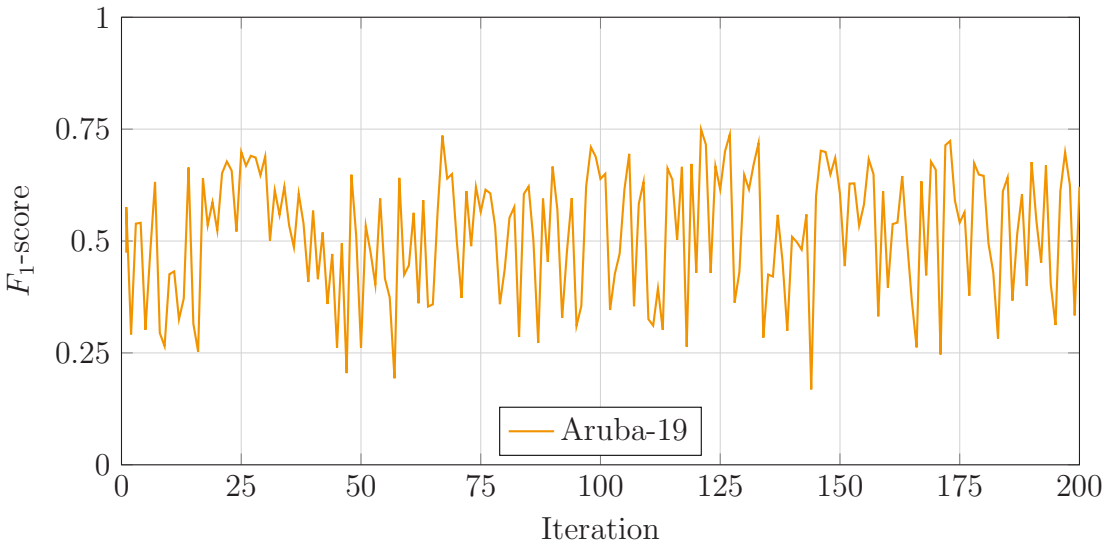


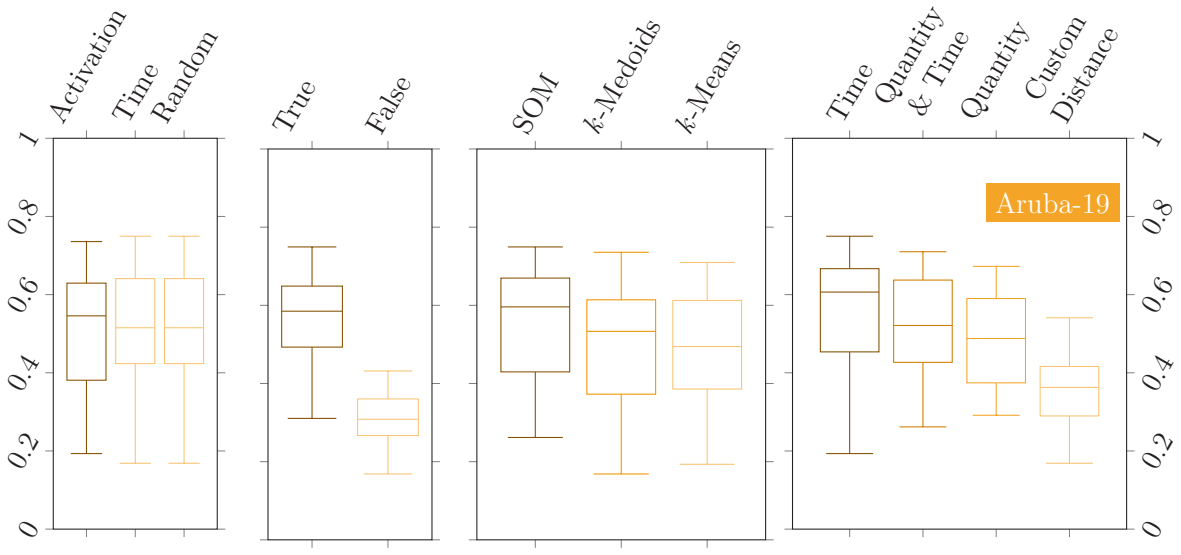
Abbildung 4.27: Funktionswert der Hyperparameter-Optimierung (Aruba)

Beide sind positiv mit dem resultierenden F_1-Wert korreliert (keine Signifikanz für Aruba-19 und die #Activation):

	Aruba-19	Kyoto-05
$corr(F_1, \#\text{Activation})$	0.0897	0.4996
$corr(F_1, \text{Duration})$	0.5042	0.3374

Die Boxplots in Abb. 4.28 zeigen, wie der F_1-Wert von den verschiedenen Hyperparametern im Aruba-19-Datensatz beeinflusst wird.

Die Beschränkung der Aktivitäten auf einen einzigen Bereich (z.B. Küche) zeigt einen durchweg niedrigeren F_1-Wert im Vergleich zu Auswertungsläufen mit erlaubten Aktivitäten in mehreren Räumen (Abb. 4.28b). Abb. 4.28c zeigt die Korrelation zwischen dem Datensatz und einem Clustering-Algorithmus. Für den Aruba-19-Datensatz übertrifft der einfache k-means-Algorithmus den komplexeren SOM-Algorithmus (Self-Organising Map). Für den Datensatz führt die Auswahl des Partitionierungsparameters *Aktivierungszeit* (Traces werden nach einer bestimmten Zeit der Sensoraktivierungsdauer abgeschnitten) zu einem hohen F_1-Score. Eine benutzerdefinierte Abstandsmetrik zwischen Traces scheint nicht immer von Vorteil zu sein, wie in 4.28d gezeigt wird. Stattdessen hängt die optimale Vektorisierung vom



(a) Trace Partition **(b)** Multi Room **(c)** Clustering Method **(d)** Vectorisation Type

Abbildung 4.28: F_1-Wert für verschiedene Parametereinstellungen in der Hyperparameter-Optimierung.

Tabelle 4.6: Ergebnisse der besten fünf von 200 Iterationen.

	F_1	Precision	Fitness	Cluste-ring	Trace Length	Number of Clusters	Day Parti-tion	Trace Partition Method
Aruba-19	0.7494	0.6540	0.8926	SOM	28 min	10	5	Time
	0.7380	0.6529	0.8709	SOM	28 min	10	5	Time
	0.7357	0.6358	0.8742	k-medoids	45 act.	16	4	Activation
	0.7187	0.6309	0.9216	SOM	28 min	8	2	Time
	0.7157	0.6158	0.8792	SOM	28 min	10	5	Time

Datensatz ab. Abb. 4.27 zeigt den Funktionswert (F_1-Score) in Abhängigkeit von der Anzahl der Iterationen.

4.2 Keine Identifikation von Entitäten

Die in diesem Abschnitt beschriebene Auswertung basiert auf der Publikation Janssen et al. (2020) und ist im Anhang C aufgeführt. Der Ansatz wurde ebenfalls anhand des öffentlich zugänglichen CASAS-Datensatzes evaluiert, der Sensor-Rohdaten aus

einer Smart Home-Umgebung enthält (Rashidi et al. 2007). Die Daten mit zwei Personen in der Testumgebung erfüllen ebenfalls die Anforderungen für das Anwenden des vorgestellten Frameworks: Sie enthalten die Zeitstempel und Standortinformationen der Sensorereignisse.

Abbildung 4.2 zeigt den Grundriss des Hauses und die Positionen der Bewegungssensoren. Für die Auswertung wurden Sensordaten 7 aufeinanderfolgender Tage (02/05–09/02/2010) aus dem Datensatz `20-Kyoto-2-Daily life, 2010-2012` extrahiert.

Die Methode wurde für verschiedene Werte von Parametern wie die Länge der Sub-Traces, die Anzahl der Cluster und das verwendete Ähnlichkeitsmaß angewendet. Es wurde eine Grid-Search-Methode verwendet, um die besten Parameterwerte für das Clustering zu ermitteln, basierend auf der durchschnittlichen kombinierten Fitness (Van der Aalst et al. 2012) und Präzision (Munoz-Gama and Carmona 2012) (F_1-Score), die für die Prozessmodelle ermittelt wurden, die für jedes Cluster von Ereignissen auf hoher Ebene entdeckt wurden. Es wurden Standard-Filtertechniken (häufigste Traces und Aktivitäten) angewendet, die im Process Mining verwendet werden, um sich auf das dominante Verhalten in jedem Cluster zu konzentrieren. Das SOM-Clustering wurde mit dem k-means-Clustering verglichen, das auf denselben Ähnlichkeitsmaßen basiert. Es wurden PM4Py und heuristicmineR[1] (Mannhardt 2023) für die Prozessfindung und -bewertung verwendet.

Nach der Entdeckung von Aktivitäten und der Extraktion von Traces, für wiederkehrendes Verhalten, das mit derselben Aktivität beginnt, wurde der Heuristic Miner angewendet, um ein Prozessmodell des menschlichen Verhaltens zu entdecken. Basierend auf der räumlichen Anordnung des Smart Home (Abbildung 4.2) wurden Traces erstellt, die mit der Aktivität *Walk entrance/stairs/storage* als Eintrittspunkt in das Haus beginnen. Der Heuristic Miner wurde ausgewählt, da zu erwarten ist, dass die Bewohner der Smart Home-Umgebung viele seltene Verhaltensweisen zeigen, die sich mit dem Heuristic Miner auffinden lassen oder gegebenenfalls durch die Filterfunktion nicht berücksichtigt werden (Augusto et al. 2018).

Abbildung 4.29 zeigt die Ergebnisse des Grid-Searchs. Die Trace-Längen wurden zwischen vier und zwölf Aktivitäten evaluiert. Kürzere Tracelängen führen im Allgemeinen zu einem besseren F_1-Score. Es sollte eine minimale Länge der Traces

[1] `https://github.com/bupaverse/heuristicsmineR`

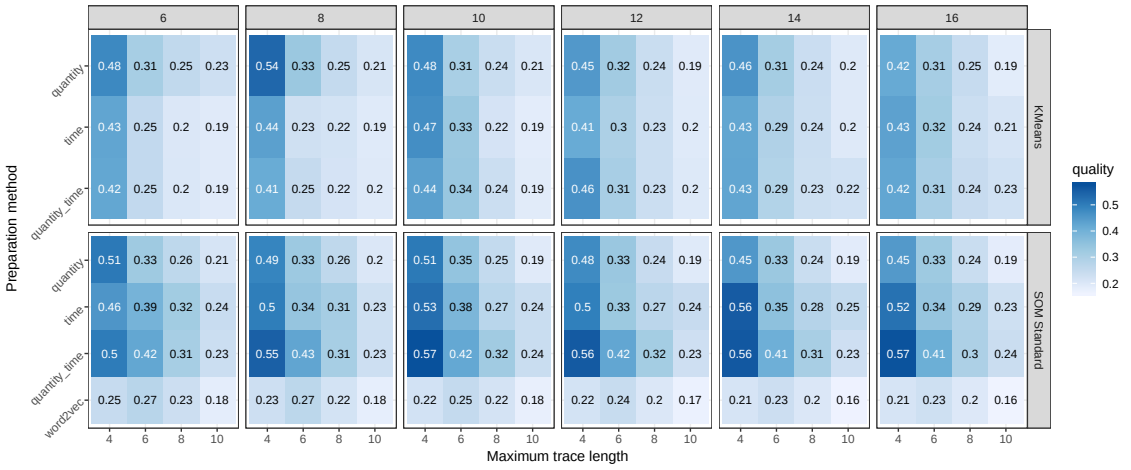


Abbildung 4.29: Durchschnittlicher F_1-Wert für Prozessmodelle, die für die Cluster auf Basis von sechs verschiedenen Clustergrößen (6–16), fünf unterschiedlichen maximalen Trace-Längen (4–12), vier Methoden zur Vektoraufbereitung und zwei Clustering-Algorithmen entdeckt wurden (Janssen et al. 2020).

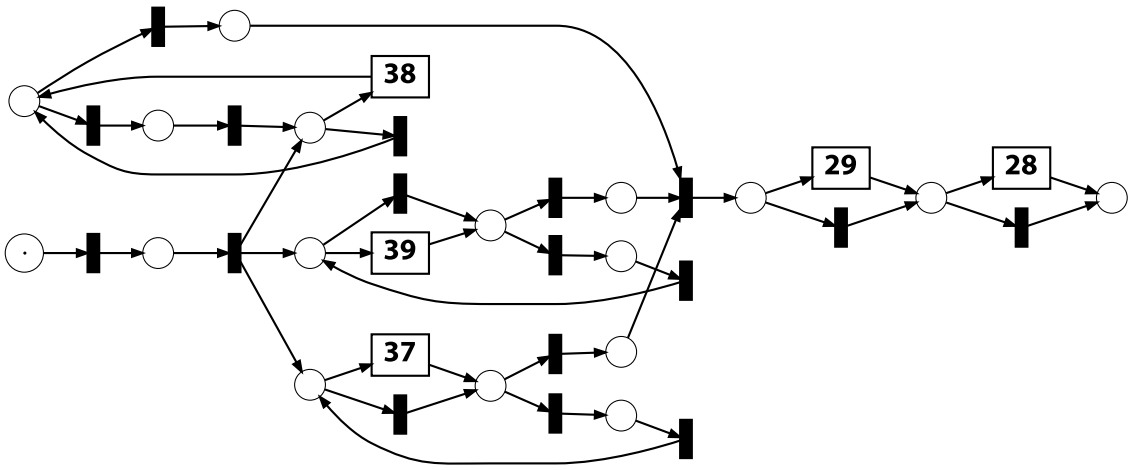


Abbildung 4.30: Beispiel eines mit dem Inductive Miner entdeckten Petri-Netzes für einen Cluster (Janssen et al. 2020).

berücksichtigt werden, da Traces, die nur aus einem einzigen Ereignis bestehen, trivialerweise zur Entdeckung von Prozessmodellen mit perfekter Fitness und Präzision führen würden. In dem vorliegenden Fall war es bei weniger als vier Ereignissen nicht möglich, eine Menge an sinnvoller Aktivitäten abzuleiten.

Das Petri-Netz dargestellt in Abbildung 4.30, ist ein vom Inductive Miner entdecktes Prozessmodell. Die drei Sensoren, die laut des Prozessmodells gleichzeitig aktiviert

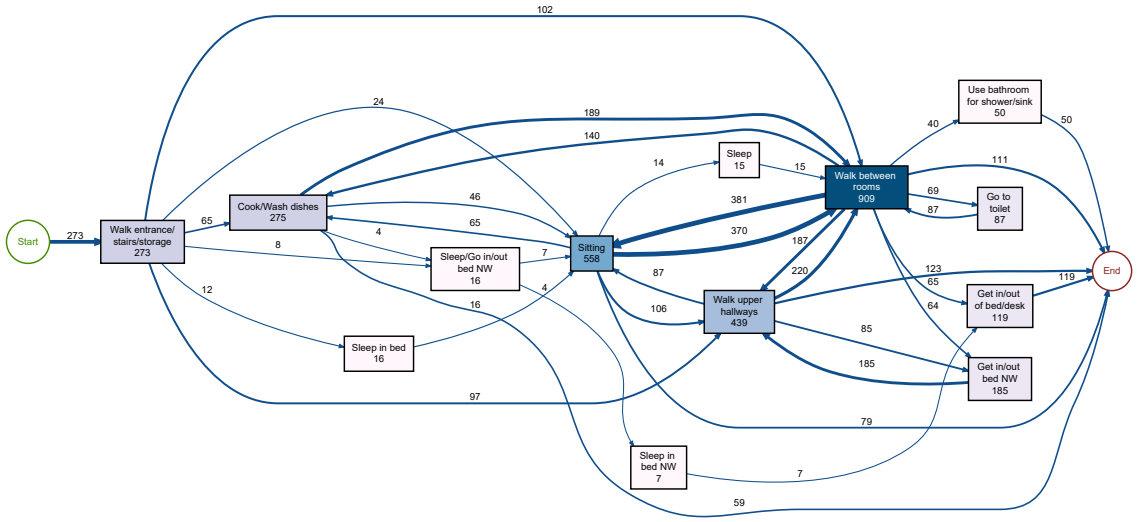


Abbildung 4.31: Causal Net entdeckt mit dem Heuristic Miner aus dem verarbeiteten Ereignislog (Janssen et al. 2020).

werden können, befinden sich im Badezimmer. Die nachfolgenden Sensoren `M29` und `M28` befinden sich im Flur mit `M28`, der am weitesten vom Badezimmer entfernt ist. Aus diesem Beispielprozessmodell lässt sich ableiten, dass sich dieser Cluster auf Aktivitäten bezieht, bei denen die Entität einige Zeit im Badezimmer verbringt und dann den Raum verlässt. Insgesamt werden ähnliche Ergebnisse für 10 und 16 Cluster mit einer Tracelänge von 4 und unter Verwendung des von uns vorgeschlagenen Vektorisierungsansatzes *quantity&time* erzielt.

Die Aktivitätsinstanzen der besten Clustering-Ergebnisse (16 Cluster) werden in Traces auf der Ebene von Prozessinstanzen gruppiert. Anschließend wurde das resultierende Ereignisprotokoll so gefiltert, dass nur Traces mit einer Länge im Bereich von 5 bis 25 Ereignissen erhalten blieben. Das Ergebnis ist ein Ereignislog mit 5898 Ereignissen, die in 273 Traces mit einer durchschnittlichen Länge von 21.6 gruppiert sind. Die Anwendung von Heuristic Miner mit einem Abhängigkeitsschwellenwert von 0.8 und einem Häufigkeitsschwellenwert von 10 liefert die in Abbildung 4.31 gezeigten Abhängigkeiten des Causal Nets. Die Aktivität im Eingangsbereich des Hauses markiert den Startpunkt des Causal Nets. Die Aktivitäten, die meist nach der Eingangsaktivität zu beobachten sind, sind *Gehen zwischen den Zimmern*, *Gehen in den oberen Fluren* und *Gehen in die Küche, um zu kochen oder das Geschirr*

abzuwaschen. Nach dem Kochen des Geschirrs kommt es häufig vor, dass die Bewohnerin zwischen den Zimmern hin und her geht, um sich zu setzen, vermutlich um im Wohnzimmer zu essen.

4.3 Synthetische Daten

Da es sich bei den bisher erwähnten Datensätzen überwiegend um ungelabelte Daten handelt, ist es sehr schwierig, die Güte des resultierenden Prozessmodells abschließend zu bewerten. Ein möglicher Lösungsansatz, um die entwickelten Algorithmen auf ihre Plausibilität hin zu überprüfen, ist die Nutzung synthetischer Daten. Die Idee hierbei ist es, einen Tagesablauf als Prozessmodell zu erstellen und dabei mögliche Sensoraktivierungen in verschiedenen Variationen zu simulieren. Das resultierende Sensor-Ereignislog wird als Input für das entwickelte Framework verwendet. Das Delta zwischen dem vom Framework erzeugten Prozessmodell und dem als Simulationsgrundlage verwendeten Prozessmodell kann genutzt werden für:

- Als Bewertungsmaß für die Effizienz der Algorithmen des Frameworks
- Für die Bewertung wie gut die Auswahl der Parameter ist

Die Publikation Zisgen et al. (2022b) aus Anhang E befasst sich mit der Generierung synthetischer Daten sowie deren Einsatz zur Validierung und Weiterentwicklung von Process Mining Algorithmen. Eine detaillierte Beschreibung der zugrunde liegenden Implementierung und des methodischen Vorgehens findet sich in den Publikationen Zisgen et al. (2022b) und Zisgen et al. (2022a).

Bei der Erzeugung dieser synthetischen Daten muss darauf geachtet werden, dass die erzeugten Daten menschlichem Verhalten möglichst nahekommen. Um realistisches menschliches Verhalten zu simulieren, dürfen die synthetischen Daten keinen starren Abläufen folgen, sondern Variationen enthalten, wie sie in der realen Welt vorkommen.

Die Idee eines solchen synthetischen Datengenerators ist es, den Tagesablauf zu Anfang mithilfe eines Petri-Netzes als Workflow-Netz zu modellieren. Durch das Verwenden von XOR-Splits (*exklusive oder* Verzweigung) bei der Modellierung können alternative Aktivitätsausführungen modelliert werden. Bei jedem Entscheidungsknoten können die Eintrittswahrscheinlichkeiten der direkt darauffolgenden Aktivitäten

Tabelle 4.7: Auszug aus einem synthetisch erstellen Ereignisprotokoll (Aktivitätsebene)

Zeitstempel	Fall-ID	Aktivität
...	...	...
2023-12-01 06:53:52	49	Aus Bett aufstehen
2023-12-01 06:59:12	49	Ins Badezimmer gehen
2023-12-01 07:02:22	49	Waschen
2023-12-01 07:23:30	49	Badezimmer verlassen
2023-12-01 07:23:30	49	Frühstückzubereitung
...	...	...

festgelegt werden, um ein nichtdeterministisches Ereignislog zu erzeugen. Außerdem kann so sichergestellt werden, dass auch selten auftretende Aktivitäten modelliert werden, ohne dass sie überproportional häufig im Ereignislog auftreten. Bei den Aktivitäten kann bestimmt werden, ob eine Aktivität zu einem bestimmten Zeitpunkt beginnen soll, bis zu einem bestimmten Zeitpunkt enden soll oder ob es keine zeitlichen Restriktionen für die Aktivität gibt. Um realistische Sensorereignislogs zu erzeugen, orientiert sich die Generierung der Ereignislogs stark an den Datensätzen des CASAS-Projekts (Cook et al. 2009). Die generierten Testdaten beinhalten sowohl Arbeitstage unter der Woche als auch arbeitsfreie Tage an Wochenenden.

Eine Beispielausgabe des Simulationsalgorithmus auf Aktivitätsebene ist exemplarisch in Tabelle 4.7 gezeigt. Die Mindestanforderungen an das Ereignislog sind mit dem Vorhandensein von Zeitstempel, Fall-ID und Aktivitätsbezeichnung gegeben.

Für das Testen des Frameworks gibt es eine weitere Variante des Simulationsalgorithmus, bei dem zusätzlich noch Sensoraktivierungen mit einbezogen werden. Um die Aktivierung von Sensoren zu simulieren, werden den einzelnen Aktivitäten Sensoren zugewiesen, die von den entsprechenden Aktivitäten ausgelöst werden. Bei der Modellierung von Bewegungssensoren ist darauf zu achten, dass jedem Aktivierungsereignis ein korrespondierendes Deaktivierungsereignis zugeordnet wird. Eine reine Statussetzung zum Aktivierungszeitpunkt ist nicht ausreichend. Die Dauer der Sensoraktivierung darf dabei nicht als konstanter Wert implementiert werden,

da dies kein realitätsnahes Verhalten abbilden würde. Stattdessen sollte sie kontextabhängig variieren und insbesondere in Abhängigkeit von der räumlichen Position des Sensors sowie der spezifischen Aktivität, die deren Auslösung verursacht, bestimmt werden.

Um die synthetisch generierten Daten noch realistischer zu machen und die Algorithmen des Frameworks auf ihre Robustheit zu überprüfen, gibt es die Möglichkeit, die erzeugten Daten mit Rauschen zu versehen. Es lassen sich sowohl die Art des Rauschens als auch die Intensität des Rauschens einstellen. Rauschen in diesem Fall würde beispielsweise eine fehlerhafte Sensorauslösung bedeuten oder ein fehlendes Deaktivierungsereignis eines Sensors.

4.4 Einordnung

Die Evaluation des vorgestellten Frameworks belegt die Funktionsfähigkeit des vorgestellten Ansatzes. Sie zeigt, dass es möglich ist, im Kontext von vernetzten Umgebungen Sensordaten zu analysieren und in strukturierte Prozessmodelle zu verarbeiten. Die Evaluation zeigt aber auch, dass die menschliche Komponente in Form eines Domänenexperten nicht zu unterschätzen ist.

Eine zusammenfassende Erkenntnis der vorgestellten Evaluation ist, dass es keine allgemein gültige Kombination von Parametern gibt. Eine optimale Parameterwahl ist immer von den jeweiligen Anwendungsfällen abhängig. Die Ergebnisqualität variiert stark, abhängig von den gewählten Datensätzen, den gewählten Parametern (Partitionierungs- und Clustering-Methoden sowie der Art der Sensorplatzierung). Für die Bewertung und Auswahl der Prozessmodelle ist die Mitwirkung eines menschlichen Domänenexperten sehr wichtig. Dies ist trotz des hohen Automatisierungsgrades des vorgestellten Frameworks der Fall. Domänenwissen ist für die Verbesserung der vom Framework erzeugten Modelle notwendig, unter anderem für:

- Interpretation von Sensormustern
- Zur Auswahl realistischer Segmentierungslängen
- Zur sinnvollen Benennung von Aktivitätsclustern

Ein Beispiel für die Notwendigkeit der menschlichen Komponente ist in den Varianten der Evaluation zu sehen, in denen gezielt der Heuristic Miner ausgewählt wurde

und der Inductive Miner von der weiteren Auswertung ausgeschlossen wurde. Die F_1-Werte in diesen Fällen übertreffen die Baseline-Ergebnisse. Dieses Beispiel zeigt, dass Entscheidungen bezüglich der Einschränkung der Parameterwahl zu einer besseren Modellqualität führen. Des Weiteren wird durch die vorgestellte Evaluation deutlich, dass Partitionierungen gemäß domänenspezifischer Tagesstrukturen oder räumliche Aktivitätsgrenzen bessere Resultate liefern als zufällige oder gleichmäßige Segmentierungen.

Die Evaluation unterstreicht eine Herangehensweise, bei der die Vorteile einer algorithmischen Optimierung und menschlicher Expertise und Intuition zusammenwirken. Das vorgestellte Framework bietet dafür eine Grundlage, indem es automatisierte Analysen erlaubt, die durch menschliche Interpretation ergänzt werden.

5 Zusammenfassung

IoT-Umgebungen erzeugen eine große Menge an Daten, die als Grundlage für neue Erkenntnisse dienen, geeignet sind. Die Verbreitung vernetzter und mit Sensoren ausgestatteter Systeme ermöglicht eine grundlegend andere Analysemöglichkeit von Daten, als dies ohne Sensoren möglich war. Hieraus ergibt sich jedoch die Herausforderung, dass diese Sensordaten häufig unstrukturiert sind und sich deutlich von den klassischen Ereignisdaten unterscheiden, mit denen herkömmliche Process-Mining-Methoden arbeiten, also jenen strukturierten Ereignislogs, die typischerweise aus betrieblichen Informationssystemen stammen. Diese Arbeit zeigt, dass eine Übertragung von Process Mining-Methoden auf unstrukturierte Sensordaten ohne Label möglich ist und in vielen Anwendungsszenarien neue Erkenntnisse erlaubt. Mit dem in dieser Dissertation vorgestellten Framework wurde ein neuartiger, domänenunabhängiger Ansatz zur automatisierten Prozessentdeckung aus Sensordaten entwickelt. Es wird gezeigt, dass Process Mining wertvolle Erkenntnisse darüber liefern kann, wie Aktivitäten in einer Smart-Home-Umgebung ablaufen, wenn aussagekräftige und eindeutige Aktivitäten aus rohen Sensorereignissen extrahiert werden. Das in dieser Arbeit vorgestellte Framework kombiniert unüberwachte Lernverfahren in Form von Clustering mit Process Mining, mit dem Ziel, Aktivitäten und Prozessmodelle aus Bewegungssensoren zu entdecken. Die Bewertung des Ansatzes erfolgte durch den Vergleich der Modellqualität für verschiedene Clustering-Techniken auf einem Datensatz, der in einem Smart Home-Szenario öffentlich verfügbar ist. Die Auswertung zeigt, dass der vorgestellte Ansatz bessere Ergebnisse liefert als die Erstellung der Prozessmodelle basierend auf zufälligen Zuordnungen (*Baseline)-Vergleich*. Um Domänenexperten größtenteils von der Prozessmodellierung zu entlasten und den Prozess der Modellfindung zu automatisieren, ist das vorgestellte Framework so konzipiert, dass ein Labeln der Aktivitäten nur am Ende der Prozessentdeckung notwendig ist.

Der Beitrag dieser Arbeit liegt in der Entwicklung eines Frameworks, das die Schritte zur Erkennung von Aktivitäten aus unstrukturierten Sensordaten umfasst: Von

der Vorverarbeitung der Sensordaten und dem Clustering ähnlicher Aktivitäten und der Prozessmodellierung bis hin zur iterativen Optimierung mittels einer Feedbackschleife. Der strukturierte Aufbau des Frameworks erlaubt es, nachträglich Methoden zu erweitern oder zu verbessern. Die Kombination von Process Mining und unüberwachtem maschinellen Lernen erlaubt es, auf manuelle Annotation weitestgehend zu verzichten. Dies ist ein entscheidender Vorteil für die Skalierung auf weitaus größere Datensätze.

Die Evaluation auf öffentlich zugänglichen Sensordatensätzen, insbesondere dem CASAS-Datensatz, zeigt die Effizienz des entwickelten Frameworks. Die entdeckten Prozessmodelle weisen eine hohe Qualität in Bezug auf Fitness und Precision auf. Die Prozessmodelle bieten eine valide Grundlage zur Erkennung typischer, sich wiederholender Verhaltensmuster. Die Erkennung von Abweichungen in Bewegungsabläufen ist ebenfalls möglich. Der halbautomatisierte Labeling-Prozess der Aktivitäten ermöglicht eine korrekte Einordnung der erkannten Aktivitäten durch Domänenexperten.

Neben den erwähnten Vorteilen weist das entwickelte Framework bestimmte Limitationen auf. Die Entitätenzuordnung basiert auf einer Heuristik und funktioniert nur zuverlässig unter der Annahme ausreichenden räumlichen Abstands sowie konsistenter Bewegungspfade der beobachteten Entitäten. Sobald Entitäten sich einander begegnen und zwischen ihnen längere Interaktionen ohne klare räumliche Trennung stattfinden, erschwert dies die eindeutige Zuordnung von Aktivitäten zu den jeweiligen Entitäten.

Die Wahl der Partitionierungsmethode bleibt abhängig von der jeweiligen Anwendungsdomäne. Zwar wird durch die Hyperparameteroptimierung ein flexibler Rahmen zur Bestimmung optimaler Segmentierungslängen der Traces geschaffen. Eine Partitionierung, die Aktivierungsabfolgen nach Kontext partitioniert, würde vermutlich deutlich bessere Ergebnisse liefern. Ein solcher Kontext wäre beispielsweise, dass Aufenthalte im Schlafzimmer tagsüber in der Regel kürzer ausfallen als während der nächtlichen Schlafenszeit. In diesem Fall wäre es sinnvoll, Traces tagsüber feiner zu segmentieren, während nachts längere Segmente gewählt werden könnten, da mit hoher Wahrscheinlichkeit die Aktivität *Schlafen* beobachtet wird.

Eine weitere Herausforderung stellt die Bewertung der Modellqualität ohne eine Ground Truth dar. Der tokenbasierte F1-Score ermöglicht zwar eine Bewertung

des Modells, es bleiben jedoch Unsicherheiten, ob die erstellten Prozessmodelle die Abläufe in den beobachteten Umgebungen tatsächlich zutreffend beschreiben.

Die vorgestellten Ergebnisse bilden das Fundament für weitere Arbeiten. So können beispielsweise weitere Sensortypen in die Auswertung mit aufgenommen werden, um die Ergebnisqualität durch mehr Daten zu erhöhen. Ein stärkerer Fokus auf die Visualisierung und Interaktion mit den Modellen könnte die Qualität der Prozessmodelle weiter steigern, da Domänenexperten interaktiv die Auswirkungen von Parameteränderungen auf die Prozessmodelle einsehen könnten.

Zusammenfassend leistet die vorliegende Arbeit einen wichtigen Beitrag, um unstrukturierte Prozesse in Smart Home Umgebungen durch die Kombination von Process Mining und (unüberwachten) Lernverfahren zu kombinieren. Es wird gezeigt, dass eine datengetriebene, weitgehend automatisierte Erkennung von Prozessen aus Sensordaten nicht nur theoretisch möglich, sondern auch praktisch umsetzbar ist. Die entwickelte Methodik ist flexibel, adaptiv und lässt sich auf eine Vielzahl von Szenarien übertragen. Damit stellt sie ein wertvolles Werkzeug zur Analyse menschlichen Verhaltens in digitalisierten, sensorbasierten Umgebungen dar – ein Forschungsfeld, dessen Relevanz mit dem Fortschreiten der Digitalisierung weiterhin enorm wichtig ist und bleiben wird.

A Publikationen

Für die eingereichten Publikationen ist in Unterkapitel A.1 der Eigenanteil zusammengefasst und in Unterkapitel A.2 werden die Publikationen zusammengefasst.

Der Zusammenhang zwischen den Publikationen ist in Abbildung A.1 dargestellt. Im dunkelblauen Bereich der Abbildungen sind die Publikationen aus **Anhang C** *Process Model Discovery from Sensor Event Data* (Janssen et al. 2020) und **Anhang D** *Process Mining on Sensor Location Event Data* (Janssen et al. 2026) dargestellt. Diese beiden Publikationen bilden den Kern dieser Dissertationsschrift. Das zuerst entwickelte und in Janssen et al. (2020) (**Anhang C**) beschriebene Framework zur Aktivitätserkennung auf Sensordaten wurde in der Publikation Janssen et al. (2026) (**Anhang D**) weiterentwickelt und ausführlicher evaluiert.

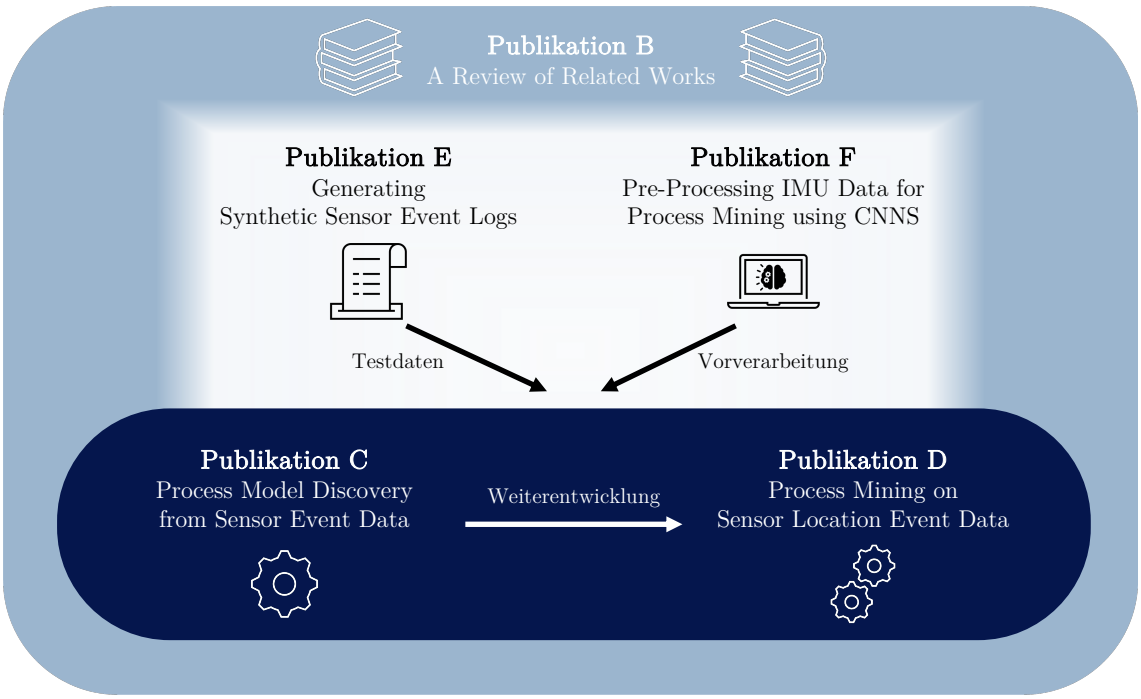


Abbildung A.1: Einordnung der Publikationen.

Um die Implementierungen aus **Anhang C** und **Anhang D** weiter zu verbessern, fehlte es an qualitativ hochwertigen Testdaten mit entsprechenden Labeln. Um dieses Problem zu adressieren, wurden eigene gelabelte synthetische Sensordaten generiert. Hieraus entstand schließlich die Publikation aus **Anhang E** *Generating Synthetic Sensor Event Logs* (Zisgen et al. 2022b).

Die Beiträge aus **Anhang C** und **Anhang D** beziehen sich auf stationäre Sensoren. Um auch mobile Sensoren abzudecken, befasst sich die Publikation *Pre-Processing IMU Data for Process Mining using CNNs* aus **Anhang F** (Polle et al. 2024) mit am Körper getragenen Sensoren. Diese Publikation beschäftigt sich mit der Vorverarbeitung von Sensordaten und kann so auch als Ergänzung zum vorgestellten Framework in **Anhang C** und **Anhang D** betrachtet werden.

Die Publikation *Process Mining on Sensor Data: A Review of Related Works* aus **Anhang B** (Brzychczy et al. 2025) beinhaltet eine ausführliche Literaturrecherche zu Sensordaten in Verbindung mit Process Mining (dargestellt in Abbildung A.1 als hellblauer Bereich). Diese Publikation deckt thematisch den *Verwandte Arbeiten*-Teil der Publikationen aus den Anhängen C, D, E und F ab.

A.1 Eigenständiger Anteil

In den folgenden Abschnitten wird der Eigenanteil an den Publikationen beschrieben (Anhänge B, C, D, E, F).

Process Mining on Sensor Data: A Review of Related Works

Der eigenständige Beitrag an der Publikation aus **Anhang B** (Brzychczy et al. 2025) umfasst die konzeptionelle Entwicklung der Forschungsfrage in enger Zusammenarbeit mit den Co-Autorinnen. Darüber hinaus habe ich eigenständig die Sichtung und Kategorisierung der Publikationen vorgenommen, die den gemeinsam festgelegten Schlüsselwörtern entsprachen. In Fällen, in denen meine Einschätzung zur Relevanz einzelner Publikationen von der Einschätzung einer Co-Autorin abwich, führte ich gemäß dem Vier-Augen-Prinzip eine Abstimmung und Diskussion mit einer dritten Co-Autorin zur finalen Entscheidungsfindung durch. Weiterhin wurden die in der Publikation verwendeten Abbildungen und Tabellen von mir erstellt. Weiterhin war

ich maßgeblich sowohl an der Erstellung des ersten Manuskriptentwurfs als auch an dessen nachfolgender Überarbeitung und Weiterentwicklung beteiligt.

Process Model Discovery from Sensor Event Data

Das Konzept zur Publikation aus **Anhang C** (Janssen et al. 2020) wurde in enger Zusammenarbeit mit Prof. Dr. Koschmider gemeinsam erarbeitet und weiterentwickelt. Die Implementierung des Frameworks wurde von mir durchgeführt. Die Evaluation des Frameworks mit Testdaten und die Erstellung der Grafiken erfolgte ebenfalls durch mich. An der Erstellung des Manuskripts war ich maßgeblich beteiligt. Die Weiterentwicklung und Überarbeitung des Manuskripts erfolgte in enger Zusammenarbeit mit den Co-Autoren.

Process Mining on Sensor Location Event Data

Die Publikation in **Anhang D** (Janssen et al. 2026) ist die direkte Weiterentwicklung der Publikation aus Anhang C (Janssen et al. 2020). Die Konzeption wurde hier ebenfalls in enger Zusammenarbeit mit Prof. Dr. Koschmider weiterentwickelt. Die Implementierung des Konzepts wurde von mir durchgeführt. Die Evaluation anhand der Testdaten wurde ebenfalls von mir durchgeführt. Ebenso erstellte ich die Grafiken und Tabellen, die unter anderem zur Darstellung der Ergebnisse dienen und die Funktionsweise des Frameworks erklären. Bei der Erstellung des ersten Entwurfs des Manuskripts war ich ebenso maßgeblich beteiligt wie bei der Weiterentwicklung des Textes.

Generating Synthetic Sensor Event Logs

Die Idee zum in **Anhang E** beschriebenen Ereignisloggenerator stammt von mir. Bei der Umsetzung der Programmierung des Generators habe ich eine beratende Rolle übernommen. Am ersten Entwurf des Manuskripts war ich maßgeblich beteiligt und habe auch die Weiterentwicklung des Textes intensiv mitgestaltet. Die Grafiken und Tabellen der Publikation wurden von mir erstellt.

Pre-Processing IMU Data for Process Mining using CNNs

Die grundlegende Konzeption und die Idee der Publikation aus **Anhang F** (Polle et al. 2024) wurden von mir entwickelt. Bei der Umsetzung der Programmierung des neuronalen Netzwerks übernahm ich eine beratende und betreuende Rolle. Zudem habe ich die Konzeption und Gestaltung der Tabellen und Grafiken übernommen. Am ersten Entwurf des Manuskripts war ich maßgeblich beteiligt und habe auch die Weiterentwicklung des Textes intensiv mitgestaltet.

A.2 Zusammenfassung der Publikationen

Im Folgenden werden die in dieser Dissertationsschrift eingebrachten Publikationen (Anhänge B, C, D, E, F) zusammengefasst.

Process Mining on Sensor Data: A Review of Related Works

In den letzten Jahren nahm das Interesse an der Anwendung von Process Mining auf unstrukturierte IoT-basierte Sensordaten stark zu. Process Mining ermöglicht es, Prozesse detailliert zu verstehen, Engpässe zu identifizieren und datengetriebene Entscheidungen zu unterstützen. Die Anwendung von Process-Mining-Methoden auf unstrukturierte IoT-Daten eröffnet neue Potenziale, um den Nutzen dieser Verfahren weiter zu steigern. Dadurch werden beispielsweise prozessbasierte Entscheidungsverfahren im Produktionskontext realisierbar. Eine direkte Anwendung von Process Mining-Verfahren ist jedoch meist nicht möglich, deshalb besteht die Herausforderung darin, rohe Sensordaten zunächst in diskrete Ereignisse zu abstrahieren und diese anschließend sinnvoll für die Analyse zu kontextualisieren.

Die Publikation aus **Anhang B** (Brzychczy et al. 2025) bietet eine umfassende systematische Literaturanalyse zur Anwendung von Process Mining auf Sensordaten. Drei zentrale Forschungsfragen werden dabei adressiert:

Forschungsfrage 1: Welche Sensortypen werden häufig verwendet und welche sind unterrepräsentiert im Bezug auf Process Mining?

Forschungsfrage 2: Welche Anwendungsbereiche und Aspekte von Process Mining werden im Kontext von Sensordaten untersucht?

Forschungsfrage 3: Welche Best Practices gibt es für die Durchführung und die Evaluierung von Process Mining auf Sensordaten?

Insgesamt wurden 36 relevante Publikationen identifiziert und analysiert. Häufig eingesetzte Sensortypen umfassen Bewegungs-, Positions-, Temperatur- und Drucksensoren, vor allem in industriellen Umgebungen, im Gesundheitswesen und in Smart-Home-Szenarien. Die meisten Arbeiten konzentrieren sich auf die Erstellung von Ereignislogs und die Prozessentdeckung. Weniger Beachtung fanden bislang Konformitätsprüfung (*Conformance Checking*) und Process Enhancement-Methoden für IoT-basierte Prozessdaten.

Die Analyse zeigte, dass häufig überwachte maschinelle Lernmethoden für die Vorverarbeitung von Sensordaten genutzt werden. Es kommen jedoch auch zunehmend unüberwachte Techniken zum Einsatz.

Basierend auf den Ergebnissen wurden folgende Empfehlungen und Forschungsrichtungen identifiziert:

- Erweiterung der Verfügbarkeit von öffentlich zugänglichen Sensordatensätzen.
- Vergleichende Studien zur Effektivität verschiedener maschineller Lernansätze für Process Mining.
- Stärkere Berücksichtigung von Konformitätsprüfung und Prozessverbesserung mithilfe von Sensordaten.
- Nutzung weiterer Sensortypen, wie etwa chemische, elektrische oder akustische Sensoren, insbesondere im industriellen Umfeld.

Die präsentierte Literaturanalyse ermöglicht erstmals eine umfassende Strukturierung des Forschungsfeldes und dient als Ausgangspunkt für künftige methodische Entwicklungen sowie praktische Anwendungen im Bereich Process Mining mit Sensordaten.

Process Model Discovery from Sensor Event Data

In der Publikation aus **Anhang C** (Janssen et al. 2020) wird ein Ansatz vorgestellt, um Prozessmodelle aus Sensordaten zu identifizieren, die typischerweise auf einem niedrigen Abstraktionsniveau (z.B. Sensorebene) vorliegen. Herkömmliche Techniken des Process Minings erwarten diskrete Ereignisdaten auf höherer Geschäftsebene, weshalb eine direkte Anwendung auf Sensordaten nicht möglich ist. Um diese Herausforderung zu bewältigen, wird eine Methode vorgestellt, die Sensorereignisse mittels unüberwachter Clustering-Verfahren diskretisiert, um daraus Aktivitäten abzuleiten.

Die Methode umfasst vier Hauptschritte:

1. **Event-Korrelation:** Rohe Sensordaten werden einzelnen Aktivitätsinstanzen zugeordnet.
2. **Aktivitätsentdeckung:** Mittels Clustering (insbesondere Self-Organising Maps (SOM) und k-Means) werden gleiche Aktivitäten gruppiert.
3. **Ereignisabstraktion:** Ereignisse werden zu Prozessaktivitäten aggregiert.
4. **Prozessmodell-Entdeckung:** Die abgeleiteten Aktivitäten werden zu einem Gesamtprozessmodell zusammengesetzt.

Evaluiert wurde der Ansatz auf dem öffentlich verfügbaren CASAS-Datensatz, welcher Bewegungsdaten aus einem Smart-Home-Szenario enthält. Dabei zeigte sich, dass insbesondere das Clustering mittels SOM bessere Ergebnisse liefert als einfache k-Means-Methoden. Die bestmögliche Modellqualität ergab sich, wenn sowohl die Anzahl als auch die Dauer der Sensoraktivierungen berücksichtigt wurden.

Trotz vielversprechender Ergebnisse weist die Methode gewisse Limitationen auf, beispielsweise bei der Erkennung und Trennung von mehreren Entitäten sowie bei der Wahl einer fixen Länge für die Aktivitätslänge. Zukünftige Forschung könnte daher variable Trace-Längen und weitere räumliche Informationen integrieren sowie komplexere Techniken zur Unterscheidung von Entitäten berücksichtigen.

Process Mining on Sensor Location Event Data

Die Publikation „Process Mining on Sensor Location Event Data“ aus **Anhang D** (Janssen et al. 2026) ist eine Weiterentwicklung der Publikation Janssen et al. (2020) und befasst sich ebenfalls mit der Herausforderung, aussagekräftige Prozessmodelle aus einfachen IoT-Sensordaten (z.B. Bewegungssensoren in Smart Homes) zu entdecken. Klassische Methoden des Process Minings stoßen hierbei auf Schwierigkeiten, da die von den Sensoren erfassten Daten oft unstrukturiert sind und zu komplexen, schwer interpretierbaren Prozessmodellen führen.

Der vorgestellte Ansatz stellt einen Lösungsansatz vor, welcher unüberwachte Lernverfahren, insbesondere Clustering, nutzt, um diese Rohdaten in abstraktere Ereignisse umzuwandeln. Die essentielle Neuerung des Ansatzes ist eine iterative Feedback-Schleife mit einer Sensitivitätsanalyse, welche automatisch die Parameter zur Aggregation und Filterung von Sensordaten optimiert. Das Clustern ähnlicher Aktivitäten erfolgt durch die Transformation von Sensordaten in Vektorform. Dies geschieht anhand verschiedener Merkmale, wie Sensor-Aktivierungsanzahl, -dauer oder einer Kombination aus beiden. Es werden Clustering-Algorithmen wie k-Means, k-Medoids und Self-Organizing Maps (SOM) verwendet, in Verbindung mit einer eigens entwickelten Distanzfunktion für sensorbasierte Raumdaten.

Kernaspekte des Ansatzes umfassen:

- Unüberwachtes Lernen (Clustering) zur Abstraktion von Sensordaten.
- Iterative Feedback-Schleife mit Sensitivitätsanalyse zur Parameteroptimierung.
- Transformation der Rohdaten in Vektorform anhand verschiedener Sensor-Merkmale.
- Verwendung verschiedener Clustering-Algorithmen: k-Means, k-Medoids und SOM.
- Eigene Distanzfunktion für raumbezogene Sensordaten.
- Bewertung der Prozessmodelle mittels F1-Score (Precision und Fitness).

Die Qualität der entdeckten Prozessmodelle wird mittels des F1-Scores (harmonisches Mittel aus Precision und Fitness) bewertet. Dieses Bewertungsverfahren dient

zugleich als Rückmeldung zur Anpassung der Cluster-Parameter, wodurch der Ansatz die Genauigkeit der entdeckten Prozessstrukturen kontinuierlich verbessert.

Evaluiert wurde die Methode anhand von zwei Datensätzen aus Smart Homes (CASAS-Projekt: Kyoto-05, Aruba-19). Die Ergebnisse zeigen, dass insbesondere längere Trace-Dauer und das Zulassen von Aktivitäten über mehrere Räume (Multi-Raum-Aktivitäten) zu besseren Modellergebnissen führen. Unterschiedliche Clustering-Methoden erzielten dabei je nach Datensatz unterschiedlich gute Ergebnisse. Die automatisierte Hyperparameteroptimierung erwies sich als besonders wertvoll, um geeignete Parameter zu bestimmen.

Abschließend zeigt das Paper überzeugend, dass der vorgeschlagene Ansatz eine deutliche Verbesserung gegenüber klassischen Verfahren bietet. Die innovative Kombination aus Process Mining und unüberwachten Lernverfahren zur automatischen Modelloptimierung eröffnet neue Perspektiven für die effiziente Analyse von IoT-Sensordaten und ist auf verschiedene IoT-Domänen übertragbar.

Die Evaluationsergebnisse zeigen, dass die vorgeschlagene Feedbackschleife die Modellqualität deutlich verbessert und es ermöglicht, aussagekräftige, höherwertige Prozessmodelle aus Sensordaten zur Standorterfassung abzuleiten.

Generating Synthetic Sensor Event Logs

„Generating Synthetic Sensor Event Logs for Process Mining“ ist die Publikation aus **Anhang E** (Zisgen et al. 2022b), die sich mit der Erstellung synthetischer Sensorereignislogs befasst.

Die Qualität und Verfügbarkeit von Ereignislogs spielen eine entscheidende Rolle für die Ergebnisqualität des Process Minings. Gerade in Szenarien mit Sensordaten oder in experimentellen Kontexten fehlen oft geeignete, reale Daten. Der vorgestellte Generator ermöglicht die Generierung synthetischer Logs, die spezifische IoT-Sensordaten enthalten können. Es ist möglich, Prozesse individuell zu modellieren und dabei die Dauer und Häufigkeit der Aktivitäten zu definieren. Zusätzlich ist es möglich, gezielt verschiedene Arten von Störungen, z. B. doppelte Ereignisse, fehlende oder inkorrekte Sensordaten hinzufügen.

Die Architektur basiert auf einer, in Python implementierten, Simulations-Engine. Diese bietet eine benutzerfreundliche Web-Oberfläche zur Prozessmodellierung mittels Petri-Netzen. Aktivitäten können mit diversen Sensoren verbunden werden. So lassen sich Prozesse, wie sie zum Beispiel im Gesundheitswesen, in vernetzten Produktionsumgebungen sowie in Smart-Home-Umgebungen vorkommen, modellieren.

Für die praktische Anwendung werden in der Publikation zwei Beispiele vorgestellt: einen Krankenhausprozess und ein Smart-Home-Szenario. In beiden Fällen werden sowohl fehlerfreie als auch mit Rauschen versehene Logs erzeugt. Besonders relevant ist dabei die Möglichkeit, durch zufällige Verteilungen realitätsnahe Abweichungen und Sonderfälle zu simulieren, was bisherige Werkzeuge nicht unterstützen.

Als Ausblick auf zukünftige Entwicklungen werden in der Publikation eine Multi-Agenten-Funktionalität mit rollenspezifischen Aufgabenverteilungen, eine realistische 3D-Modellierungsumgebung sowie eine bessere Reproduzierbarkeit der Simulationen durch eine Zufallssteuerung mit festlegbarem Startwert (*Seed Key*) vorgestellt. Ziel ist es, das Werkzeug als Plattform für die Generierung synthetischer Sensorereignislogs nutzen zu können und dadurch die Forschung und Anwendung im Bereich Process Mining, insbesondere in Verbindung mit Sensordaten, voranzubringen.

Pre-Processing IMU Data for Process Mining using CNNs

Die in **Anhang F** (Polle et al. 2024) vorgestellte Publikation beschäftigt sich damit, unstrukturierte Daten, aufgezeichnet von inertialen Messeinheiten (Inertial Measurement Units – IMUs) für Prozess-Mining-Techniken nutzbar zu machen. In diesem Kontext wird Prozess Mining verwendet, um anhand der aufgezeichneten Daten Prozesse zu rekonstruieren, Abweichungen zu erkennen und zukünftige Aktivitäten vorherzusagen. Typischerweise basieren solche Verfahren auf strukturierten Ereignisdaten, etwa aus IT-Systemen. Da es sich aber um Bewegungsdaten aus Sensoren wie Accelerometern, Gyroskopen und Magnetometern handelt, muss eine Methode entwickelt werden, wie auf diesen kontinuierlich aufgezeichneten, unstrukturierten Sensordaten Process Mining-Verfahren angewendet werden können. Dies erfordert umfassende Vorverarbeitungsschritte, bevor es möglich ist, sie für das Process-Mining nutzen zu können.

Um die oben genannten Herausforderungen zu bewältigen, stellt diese Publikation einen systematischen Ansatz zur Vorverarbeitung von IMU-Daten zur Generierung von Ereignislogs vor. Die entwickelte Methodik umfasst mehrere Schritte, die in einer Pipeline zur Vorverarbeitung zusammengefasst werden:

Datenerfassung: Die Datenerfassung erfolgt mittels am Körper getragener Sensoren wie Smartwatches oder speziellen Bewegungssensoren.

Datenbereinigung: In diesem Schritt wird Rauschen reduziert. Seltene oder sehr ähnliche Aktivitäten werden zusammengefasst bzw. entfernt.

Segmentierung: Anschließend werden die kontinuierlichen Rohdaten segmentiert. Hierfür werden Zeitfenster mit konstanter Länge gebildet, wobei kurze Segmente mit sogenannten „Zero-Padding"-Techniken auf eine einheitliche Länge gebracht werden. Diese einheitlichen Fenster dienen als Eingabe für Convolutional Neural Networks (CNNs), die schließlich die Zuordnung der Fenster zu spezifischen Alltagsaktivitäten ermöglichen.

In der Publikation werden mehrere Varianten der CNN-basierten Aktivitätserkennung evaluiert und verglichen. Zum Einsatz kamen dabei die beiden CNN-Modelle „MiniRocket" und „MultiRocket", die sich durch ihre Effizienz bei der Verarbeitung von Zeitreihen auszeichnen. Es wurden außerdem verschiedene Segmentierungsstrategien betrachtet:

- Es wurden Zeitfenster untersucht, deren genaue Grenzen bereits bekannt waren, zum Beispiel durch manuelle Annotation
- Es wurden Zeitfenster unbekannter Länge untersucht. Diese wurden mit automatischen Verfahren identifiziert. Für diese wurden sowohl eine Methode mit festen Sliding-Windows, als auch eine Change-Point-Erkennung untersucht.

Für die Evaluation wurde ein umfangreicher Datensatz herangezogen, der Bewegungsdaten von insgesamt 109 Probanden umfasst. Die Daten wurden im Labor aufgenommen und beinhalten die Ausführung alltäglicher Tätigkeiten wie Gehen, Aufstehen, Hinsetzen, das Decken eines Tisches und das Greifen oder Umplatzieren von Objekten. Insgesamt umfasst der Datensatz 3392 annotierte Aktivitätsinstanzen. Selten auftretende Aktivitäten wurden aus Gründen der Modellgenauigkeit von der Analyse ausgeschlossen.

Die Ergebnisse der Evaluation zeigen, dass CNN-basierte Ansätze grundsätzlich geeignet sind, IMU-Daten zuverlässig in strukturierte Ereignislogs umzuwandeln. Insbesondere bei bekannten Zeitfenstern erzielen CNNs sehr gute Ergebnisse. In Situationen, in denen die genauen Aktivitätsgrenzen unbekannt sind, bewährt sich besonders der Ansatz mit fixen Sliding-Windows gegenüber der Change-Point-Detection. Vor allem monotone Bewegungen wie Gehen oder Positionswechsel („Sit-to-Stand“ und „Stand-to-Sit“) wurden durch diese Methode gut identifiziert. Aktivitäten mit komplexeren Bewegungsabläufen (wie das Decken eines Tisches) konnten dagegen mithilfe der Change-Point-Detection besser erkannt werden.

Eine Erkenntnis der Evaluation ist, dass kleinere Zeitfenster zu einer höheren Erkennungsgenauigkeit führen als größere Fenster. Ebenfalls wurde festgestellt, dass die Modellperformance stark von der Auftrittshäufigkeit der jeweiligen Aktivitäten abhängt: Häufig ausgeführte Aktivitäten werden zuverlässiger erkannt als weniger häufig auftretende.

Als Ausblick wird vorgeschlagen, die Methode auf komplexere und realitätsnähere Szenarien zu übertragen. Hierzu gehören insbesondere Messungen in Umgebungen außerhalb des Labors sowie die Integration weiterer Sensorinformationen, wie Ortsdaten. Darüber hinaus sollen weitere Ansätze zur Optimierung der CNN-Modelle untersucht werden, etwa die Optimierung von Hyperparametern oder der Vergleich mit anderen Machine-Learning-Modellen.

B Process Mining on Sensor Data: A Review of Related Works

Edyta Brzychczy	AGH University of Krakow, Poland
Milda Aleknonytė-Resch	Kiel University, Germany
Dominik Janssen	University of Bayreuth, Germany
Agnes Koschmider	University of Bayreuth, Germany

Process mining is an efficient technique that combines data analysis and behavioural process aspects to uncover end-to-end processes from data. Recently, the application of process mining on unstructured data has become popular. Particularly, sensor data from IoT-based systems allow process mining to uncover novel insights that can be used to identify bottlenecks in the process and support decision-making. However, the application of process mining requires bridging challenges. First, (raw) sensor data must be abstracted into discrete events to be useful for process mining. Second, meaningful events must be distilled from the abstracted events, fulfilling the purpose of the analysis. In this paper, a comprehensive literature study is conducted to understand the field of process mining for sensor data. The literature search was guided by three research questions: (1) what are common and underrepresented sensor types for process mining, (2) which aspects of process mining are covered on sensor data, and (3) what are the best practices to improve the understanding, design, and evaluation of process mining on sensor data. A total of 36 related papers were identified, which were then used as a foundation to structure the field of process mining on sensor data and provide recommendations and future research directions. The findings serve as a starting point for designing new techniques, enhancing the dissemination of related approaches, and identifying research gaps in process mining on sensor data.

Published in	Knowledge and Information Systems
Citation	Brzychczy et al. (2025)

C Process Model Discovery from Sensor Event Data

Dominik Janssen	Kiel University, Germany
Felix Mannhardt	SINTEF Digital, Norway
Agnes Koschmider	Kiel University, Germany
Sebastiaan J. van Zelst	RWTH Aachen University, Germany

Virtually all techniques, developed in the area of process mining, assume the input event data to be discrete, and, at a relatively high level (i.e., close to the business-level). However, in many cases, the event data generated during the execution of a process is at a much lower level of abstraction, e.g., sensor data. Hence, in this paper, we present a novel technique that allows us to translate sensor data into higher-level, discrete event data, thus enabling existing process mining techniques to work on data tracked at a sensory level. Our technique discretises the observed sensor data into activities by applying unsupervised learning in the form of clustering. Furthermore, we refine the observed sequences by deducing imperative sub-models for the observed discretised data, i.e., allowing us to identify concurrency and interleaving within the data. We evaluated the approach by comparing the obtained model quality for several clustering techniques on a publicly available data-set in a smart home scenario. Our results show that applying our framework combined with a clustering technique yields results on data that otherwise would not be suitable for process discovery.

Published in	Process Mining Workshops. ICPM 2020. LNBIP, vol 406
Presented at	1st Int. Workshop on Event Data and Behavioral Analytics
Citation	Janssen et al. (2020)

D Process Mining on Sensor Location Event Data

Dominik Janssen	University of Bayreuth, Germany
Agnes Koschmider	University of Bayreuth, Germany
Felix Mannhardt	Eindhoven University of Technology, The Netherlands

The increasing amount of IoT sensor event data in domains like smart cities or logistics demands efficient analysis techniques that give valuable and new insights from data. Process mining promises to discover valuable knowledge from data. However, the event data tracked from IoT devices is at a much lower level, on which a direct application of process mining yields few insights. This paper suggests an approach for process discovery on sensor location event data in single occupation settings by leveraging unsupervised learning in the form of clustering to raise the abstraction level of events. The main contribution is a feedback loop with a sensitivity analysis that measures how sensitive the discovered process model is to changes when processing and aggregating IoT sensor location event data in a (semi)automated manner. Our evaluation results show that the feedback loop automatically improves model quality considerably and can obtain high-level processes from sensor location event data.

Published in	Internet of Things Meets Business Process Management
Presented at	7th International Workshop on Business Processes Meet IoT
Citation	Janssen et al. (2026)

E Generating Synthetic Sensor Event Logs for Process Mining

Yorck Zisgen	Kiel University, Germany
Dominik Janssen	Kiel University, Germany
Agnes Koschmider	Kiel University, Germany

Process mining has gained significant practical usefulness in diverse domains. The input of process mining is an event log, tracking the execution of activities that can be mapped onto a business processes. Thus, the availability and quality of event logs significantly impact the process mining result. The use of process mining in novel use cases or experimental settings is often hampered because no appropriate event logs are available. This paper presents a tool to generate synthetic (sensor) event logs. Compared to existing synthetic log generator tools, the IoT process log generator produces data in a non-deterministic way. Users can add noise in a controlled manner and might enhance the processes with IoT data. In this way, the tool allows generating synthetic data for IoT environments that can be individually configured. Our tool makes a contribution towards an increased use of process mining in settings relying on (IoT) sensor event data.

Published in	Intelligent Information Systems. CAiSE 2022. LNBIP, vol 452
Presented at	34rd Int. Conf. on Advanced Information Systems Engineering
Citation	Zisgen et al. (2022b)

F Pre-Processing Inertial Measurement Unit-based Data for Process Mining using Convolutional Neural Networks

Daniel Polle	University of Bayreuth, Germany
Milda Aleknonytė-Resch	Kiel University, Germany
Dominik Janssen	University of Bayreuth, Germany
Clint Hansen	Kiel University, Germany
Elke Warmerdam	Saarland University, Germany
Walter Maetzler	Kiel University, Germany
Agnes Koschmider	University of Bayreuth, Germany

The analysis of inertial measurement unit (IMU)-based data allows tracking human behavior, detecting anomalies, and predicting human activity changes. As IMU-based data is unstructured and continuous, the application of process mining could provide additional insights into the underlying human performance. Therefore, the data has to be efficiently pre-processed in order to be used by process mining algorithms. This paper presents an approach to convert IMU-based data into structured event data for process mining. Particularly, the approach relies on methods for time-series segmentation and convolutional neural networks. In this way, activities of daily living can be identified from the unstructured data. The evaluation results show that convolutional neural networks are suitable for discovering activities when window sizes are previously known and have low cutoff values. The combination with a fixed sliding window approach for unknown window sizes appears superior.

Published in	Wirtschaftsinformatik 2024 Proceedings. 68
Presented at	19. Internationale Tagung Wirtschaftsinformatik
Citation	Polle et al. (2024)

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Eigene Veröffentlichungen

Konferenzbeiträge

Dominik Janssen, Felix Mannhardt, Agnes Koschmider, and Sebastiaan J van Zelst. Process model discovery from sensor event data. In *International Conference on Process Mining*, pages 69–81. Springer, 2020.

Dominik Janssen, Agnes Koschmider, and Felix Mannhardt. Process mining on sensor location event data. In Giancarlo Fortino and Massimo Mecella, editors, *Internet of Things Meets Business Process Management*, Internet of Things, pages –. Springer Nature Switzerland, 2026. ISBN 978-3-031-90746-3. doi: 10.1007/978-3-031-90746-3_XX. In press. Forthcoming December 2025. Part of the Internet of Things series (ISSN 2199-1073, eISSN 2199-1081).

Agnes Koschmider, Dominik Janssen, and Felix Mannhardt. Framework for process discovery from sensor data. In *10th International Workshop on Enterprise Modeling and Information Systems Architectures, EMISA 2020*, pages 32–38. CEUR-WS. org, 2020.

Agnes Koschmider, Milda Aleknonytė-Resch, Frederik Fonger, Christian Imenkamp, Arvid Lepsien, Kaan Apaydin, Dominik Janssen, Dominic Langhammer, Tobias Ziolkowski, and Yorck Zisgen. Process mining for unstructured data: Challenges and research directions. In *Modellierung 2024*, pages 119–136. Gesellschaft für Informatik e.V., Bonn, 2024. ISBN 978-3-88579-742-5. doi: 10.18420/modellierung2024_012.

Saskia Nuñez von Voigt, Stephan A Fahrenkrog-Petersen, Dominik Janssen, Agnes Koschmider, Florian Tschorsch, Felix Mannhardt, Olaf Landsiedel, and Matthias Weidlich. Quantifying the re-identification risk of event logs for process mining: Empiricial evaluation paper. In *Advanced Information Systems Engineering: 32nd*

International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings 32, pages 252–267. Springer, 2020.

Daniel Polle, Milda Aleknonyté-Resch, Dominik Janssen, Clint Hansen, Elke Warmerdam, Walter Maetzler, and Agnes Koschmider. Pre-processing inertial measurement unit-based data for process mining using convolutional neural networks. In *Wirtschaftsinformatik 2024 Proceedings 68*. AIS, 2024.

Yorck Zisgen, Dominik Janssen, Christian Imenkamp, and Agnes Koschmider. Modellierungsumgebung zur erzeugung synthetischer ereignisprotokolle für das process mining. In *Modellierung 2022 Satellite Events*, pages 256–265. Gesellschaft für Informatik eV, 2022a.

Yorck Zisgen, Dominik Janssen, and Agnes Koschmider. Generating synthetic sensor event logs for process mining. In *International Conference on Advanced Information Systems Engineering*, pages 130–137. Springer, 2022b.

Journalartikel

Edyta Brzychczy, Milda Aleknonytė-Resch, Dominik Janssen, and Agnes Koschmider. Process mining on sensor data: a review of related works. *Knowledge and Information Systems*, pages 1–34, 2025.

Literaturverzeichnis

Rafael Accorsi, Meike Ullrich, and W van der Aalst. Aktuelles schlagwort: Process mining. *Informatik Spektrum*, 35(5):354–359, 2012.

Amirah Alharbi, Andy Bulpitt, and Owen A Johnson. Towards unsupervised detection of process models in healthcare. In *Building Continents of Knowledge in Oceans of Data: The Future of Co-Created eHealth*, pages 381–385. IOS Press, 2018.

Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs: Review and benchmark. *IEEE transactions on knowledge and data engineering*, 31(4):686–705, 2018.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

Alessandro Berti and Wil MP van der Aalst. Reviving token-based replay: Increasing speed while improving diagnostics. In *ATAED@ Petri Nets/ACSD*, pages 87–103, 2019.

Alessandro Berti, Sebastiaan J van Zelst, and Wil van der Aalst. Pm4py: bridging the gap between process-and data science. *arXiv preprint arXiv:1905.06169*, 2019.

Alessandro Berti, Sebastiaan van Zelst, and Daniel Schuster. Pm4py: A process mining library for python. *Software Impacts*, 17:100556, 2023.

James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences*, 10(2-3):191–203, 1984.

Alejandro Bogarín Vega, Rebeca Cerezo Menéndez, Cristobal Romero, et al. Discovering learning processes using inductive miner: A case study with learning management systems (lmss). *Psicothema*, 2018.

Edyta Brzychczy and Agnieszka Trzcionkowska. Process-oriented approach for analysis of sensor data from longwall monitoring system. In *Intelligent Systems in Production Engineering and Maintenance*, pages 611–621. Springer, 2019.

Joos CAM Buijs, Boudewijn F van Dongen, and Wil MP van Der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Confederated International ConferencesÖn the Move to Meaningful Internet Systems"*, pages 305–322. Springer, 2012.

Marco Cameranesi, Claudia Diamantini, Alex Mircoli, Domenico Potena, and Emanuele Storti. Extraction of user daily behavior from home sensors through process discovery. *IEEE Internet of Things Journal*, 7(9):8440–8450, 2020.

Josep Carmona, Boudewijn van Dongen, Andreas Solti, and Matthias Weidlich. Conformance checking. *Switzerland: Springer.[Google Scholar]*, 56, 2018.

Sungjoon Choi, Eunwoo Kim, and Songhwai Oh. Human behavior prediction for smart homes using deep learning. In *2013 IEEE RO-MAN*, pages 173–179. IEEE, 2013.

Mohamed-Amine Choukou, Taylor Shortly, Nicole Leclerc, Derek Freier, Genevieve Lessard, Louise Demers, and Claudine Auger. Evaluating the acceptance of ambient assisted living technology (aalt) in rehabilitation: A scoping review. *International journal of medical informatics*, 150:104461, 2021.

Diane Cook, Maureen Schmitter-Edgecombe, Aaron Crandall, Chad Sanders, and Brian Thomas. Collecting and disseminating smart home sensor data in the casas project. In *Proceedings of the CHI workshop on developing shared home behavior datasets to advance HCI and ubiquitous computing research*, pages 1–7, 2009.

Diane J Cook and Maureen Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. *Methods of information in medicine*, 48(05):480–485, 2009.

Jochen De Weerdt, Seppe Vanden Broucke, Jan Vanthienen, and Bart Baesens. Active trace clustering for improved process discovery. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2708–2720, 2013.

George Demiris, Brian K Hensel, Marjorie Skubic, and Marilyn Rantz. Senior residents' perceived need of and preferences for "smart home" sensor technologies. *International journal of technology assessment in health care*, 24(1):120–124, 2008.

Stefan Dernbach, Barnan Das, Narayanan C Krishnan, Brian L Thomas, and Diane J Cook. Simple and complex activity recognition through smart phones. In *2012 eighth international conference on intelligent environments*, pages 214–221. IEEE, 2012.

Claudia Diamantini, Laura Genga, and Domenico Potena. Behavioral process mining for unstructured processes. *Journal of Intelligent Information Systems*, 47(1):5–32, 2016.

Alexander Diete, Timo Sztyler, Lydia Weiland, and Heiner Stuckenschmidt. Improving motion-based activity recognition with ego-centric vision. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 488–491. IEEE, 2018.

Marcella Dimaggio, Francesco Leotta, Massimo Mecella, and Daniele Sora. Process-based habit mining: Experiments and techniques. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 145–152. IEEE, 2016.

Onur Dogan, Antonio Martinez-Millana, Eric Rojas, Marcos Sepúlveda, Jorge Munoz-Gama, Vicente Traver, and Carlos Fernandez-Llatas. Individual behavior modeling with sensors using process mining. *Electronics*, 8(7):766, 2019.

Onur Dogan, Basar Oztaysi, and Carlos Fernandez-Llatas. Segmentation of indoor customer paths using intuitionistic fuzzy clustering: Process mining visualization. *Journal of Intelligent & Fuzzy Systems*, 38(1):675–684, 2020.

Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6): 345–345, 1962.

Claudio Gallicchio, Alessio Micheli, et al. Experimental analysis of deep echo state networks for ambient assisted living. *AI* AAL@ AI* IA*, 2061:44–57, 2017.

Christian W Günther and Anne Rozinat. Disco: Discover your processes. In *Demonstration Track of the 10th International Conference on Business Process Management, BPM Demos 2012*, pages 40–44. CEUR-WS. org, 2012.

Christian W Günther and Wil MP Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007.

Ping Guo and Zhenjiang Miao. Multi-person activity recognition through hierarchical and observation decomposed HMM. In *2010 IEEE International Conference on Multimedia and Expo*, pages 143–148. IEEE, IEEE, July 2010. doi: 10.1109/icme.2010.5582559.

Esmita P Gupta. Process mining a comparative study. *International Journal of Advanced Research in Computer and Communications Engineering*, 3(11):5, 2014.

Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28 (1):100–108, 1979.

Elvis Hozdić. Smart factory for industry 4.0: A review. *International Journal of Modern Manufacturing Technologies*, 7(1):28–35, 2015.

Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

Christian Janiesch, Agnes Koschmider, Massimo Mecella, Barbara Weber, Andrea Burattin, Claudio Di Ciccio, Giancarlo Fortino, Avigdor Gal, Udo Kannengiesser, Francesco Leotta, et al. The internet of things meets business process management: a manifesto. *IEEE Systems, Man, and Cybernetics Magazine*, 6(4):34–44, 2020.

Yin Jie, Ji Yong Pei, Li Jun, Guo Yun, and Xu Wei. Smart home system based on iot technologies. In *2013 International conference on computational and information sciences*, pages 1789–1791. IEEE, 2013.

Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

Ulrich Knauer and Kolja Knauer. *Algebraic graph theory: morphisms, monoids and matrices*, volume 41. Walter de Gruyter GmbH & Co KG, 2019.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

Agnes Koschmider and Daniel Siqueira Vidal Moreira. Change detection in event logs by clustering. In *On the Move to Meaningful Internet Systems. OTM 2018 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part I*, pages 643–660. Springer, 2018.

Agnes Koschmider, Felix Mannhardt, and Tobias Heuser. On the contextualization of event-activity mappings. In *Business Process Management Workshops: BPM 2018 International Workshops, Sydney, NSW, Australia, September 9-14, 2018, Revised Papers 16*, pages 445–457. Springer, 2019.

Marcello La Rosa, Hajo A Reijers, Wil MP Van Der Aalst, Remco M Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. Apromore: An advanced process model repository. *Expert Systems with Applications*, 38(6):7029–7040, 2011.

Grégoire S Larue, Andry Rakotonirainy, and Anthony N Pettitt. Predicting reduced driver alertness on monotonous highways. *IEEE Pervasive Computing*, 14(2):78–85, 2015.

Ralf Laue, Agnes Koschmider, and Dirk Fahland. *Prozessmanagement und Process-Mining: Grundlagen*. Walter de Gruyter GmbH & Co KG, 2020.

Sander JJ Leemans. Robust process mining with guarantees. In *BPM (Dissertation/Demos/Industry)*, pages 46–50. Springer, 2018.

Sander JJ Leemans, Dirk Fahland, and Wil MP Van Der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *International conference on applications and theory of Petri nets and concurrency*, pages 311–329. Springer, 2013.

Francesco Leotta, Massimo Mecella, and Jan Mendling. Applying process mining to smart spaces: Perspectives and research challenges. In *International conference on advanced information systems engineering*, pages 298–304. Springer, 2015.

Francesco Leotta, Massimo Mecella, and Daniele Sora. Visual process maps: A visualization tool for discovering habits in smart homes. *Journal of Ambient Intelligence and Humanized Computing*, 11(5):1997–2025, 2020.

Xixi Lu, Dirk Fahland, Frank JHM van den Biggelaar, and Wil MP van der Aalst. Handling duplicated tasks in process discovery by refining event labels. In *International Conference on Business Process Management*, pages 90–107. Springer, 2016.

Felix Mannhardt. heuristicsminer: discovery of process models with the heuristics miner. 2023.

Felix Mannhardt and Niek Tax. Unsupervised event abstraction using pattern abstraction and local process models. *arXiv preprint arXiv:1704.03520*, 2017.

Wes McKinney et al. Data structures for statistical computing in python. *SciPy*, 445(1):51–56, 2010.

Ved Prakash Mishra, Joanita Dsouza, and Laura Elizabeth. Analysis and comparison of process mining algorithms with application of process mining in intrusion detection system. In *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 613–617. IEEE, 2018.

Jorge Munoz-Gama and Josep Carmona. A general framework for precision checking. *International Journal of Innovative Computing, Information and Control (IJICIC)*, 8(7):5317–5339, 2012.

Ehsan Nazerfard. Temporal features and relations discovery of activities from sensor data. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–16, 2018.

Mirko Nentwig and Marc Stamminger. A method for the reproduction of vehicle test drives for the simulation based evaluation of image processing algorithms. In *13th International IEEE conference on intelligent transportation systems*, pages 1307–1312. IEEE, 2010.

HD Nguyen, Kim Phuc Tran, Xianyi Zeng, Ludovic Koehl, and Guillaume Tartare. Wearable sensor data based human activity recognition using machine learning: a new approach. *arXiv preprint arXiv:1905.03809*, 2019.

Andreas Oberweis. *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Springer-Verlag, 2013.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Carl Adam Petri. Kommunikation mit automaten. 1962.

Norbert Pohlmann. Chancen und risiken von smart home. *Datenschutz und Datensicherheit-DuD*, 45(2):95–101, 2021.

Rasika S Ransing and Manita Rajput. Smart home for elderly care, based on wireless sensor network. In *2015 International Conference on Nascent Technologies in the Engineering Field (ICNTE)*, pages 1–5. IEEE, 2015.

Parisa Rashidi, G Michael Youngblood, Diane J Cook, and Sajal K Das. Inhabitant guidance of smart environments. In *Human-Computer Interaction. Interaction Platforms and Techniques: 12th International Conference, HCI International 2007, Beijing, China, July 22-27, 2007, Proceedings, Part II 12*, pages 910–919. Springer, 2007.

Michael Schiefer. Smart home definition and security threats. In *2015 ninth international conference on IT security incident management & IT forensics*, pages 114–118. IEEE, 2015.

Ronny Seiger, Marco Franceschetti, and Barbara Weber. An interactive method for detection of process activity executions from iot data. *Future Internet*, 15(2):77, 2023.

Arik Senderovich, Andreas Rogge-Solti, Avigdor Gal, Jan Mendling, and Avishai Mandelbaum. The road from sensor data to process instances via interaction mining. In *Int. Conference on Advanced Information Systems Engineering*, pages 257–273. Springer, 2016.

Nagender Kumar Suryadevara, Subhas C Mukhopadhyay, Ruili Wang, and RK Rayudu. Forecasting the behavior of an elderly using wireless sensors data in a smart home. *Engineering Applications of Artificial Intelligence*, 26(10):2641–2652, 2013.

Timo Sztyler. *Sensor-based human activity recognition: Overcoming issues in a real world setting*. Universitaet Mannheim (Germany), 2019.

Timo Sztyler, Josep Carmona, Johanna Völker, and Heiner Stuckenschmidt. Self-tracking reloaded: applying process mining to personalized health care from labeled sensor data. *Transactions on Petri nets and other models of concurrency XI*, pages 160–180, 2016.

Niek Tax, Xixi Lu, Natalia Sidorova, Dirk Fahland, and Wil MP van der Aalst. The imprecisions of precision measures in process mining. *Information Processing Letters*, 135:1–8, 2018a.

Niek Tax, Natalia Sidorova, Reinder Haakma, and W van der Aalst. Mining process model descriptions of daily life through event abstraction. In *Intelligent Systems and Applications: Extended and Selected Results from the SAI Intelligent Systems Conference (IntelliSys) 2016*, pages 83–104. Springer, 2018b.

Niek Tax, Emin Alasgarov, Natalia Sidorova, Reinder Haakma, and Wil MP van der Aalst. Generating time-based label refinements to discover more precise process models. *Journal of Ambient Intelligence and Smart Environments*, 11(2):165–182, 2019.

Rahila Umer, Teo Susnjak, Anuradha Mathrani, and Suriadi Suriadi. On predicting academic performance with process mining in learning analytics. *Journal of Research in Innovative Teaching & Learning*, 2017.

Han Van Der Aa, Henrik Leopold, and Hajo A Reijers. Checking process compliance on the basis of uncertain event-to-activity mappings. In *Advanced Information Systems Engineering: 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings 29*, pages 79–93. Springer, 2017.

Wil Van Der Aalst. Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 3(2):1–17, 2012a.

Wil Van Der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, 2012b.

Wil van der Aalst. Process mining: The missing link. *Process mining: data science in action*, pages 25–52, 2016a.

Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE transactions on knowledge and data engineering*, 16(9):1128–1142, 2004.

Wil Van Der Aalst, Arya Adriansyah, and Boudewijn Van Dongen. Causal nets: A modeling language tailored towards process discovery. In *International conference on concurrency theory*, pages 28–42. Springer, 2011.

Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I 9*, pages 169–194. Springer, 2012.

Wil Van der Aalst, Arya Adriansyah, and Boudewijn Van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016b. ISBN 978-3-662-49850-7. doi: 10.1007/978-3-662-49851-4. URL `https://doi.org/10.1007/978-3-662-49851-4`.

Wil MP Van der Aalst. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets*, pages 407–426. Springer, 1997.

Wil MP van der Aalst. Process discovery: Capturing the invisible. *IEEE Computational Intelligence Magazine*, 5(1):28–41, 2010.

Wil MP van der Aalst. A practitioner's guide to process mining: limitations of the directly-follows graph, 2019.

Boudewijn F Van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP van Der Aalst. The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. Proceedings 26*, pages 444–454. Springer, 2005.

Maikel L Van Eck, Natalia Sidorova, and Wil MP Van der Aalst. Enabling process mining on sensor data from smart products. In *2016 IEEE tenth international conference on research challenges in information science (RCIS)*, pages 1–12. IEEE, 2016.

Kees M van Hee, Natalia Sidorova, and Jan Martijn van der Werf. Business process modeling using petri nets. In *Transactions on Petri Nets and Other Models of Concurrency VII*, pages 116–161. Springer, 2013.

Seppe KLM vanden Broucke and Jochen De Weerdt. Fodina: a robust and flexible heuristic process discovery technique. *decision support systems*, 100:109–118, 2017.

Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on neural networks*, 11(3):586–600, 2000.

Aiguo Wang, Guilin Chen, Cuijuan Shang, Miaofei Zhang, and Li Liu. Human activity recognition in a smart home environment with stacked denoising autoencoders. In *Web-Age Information Management: WAIM 2016 International Workshops, MWDA, SDMMW, and SemiBDMA, Nanchang, China, June 3-5, 2016, Revised Selected Papers 17*, pages 29–40. Springer, 2016.

Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern recognition letters*, 119: 3–11, 2019.

Anton JMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. Process mining with the heuristicsminer algorithm. 2006.

Mi Zhang and Alexander A Sawchuk. Motion primitive-based human activity recognition using a bag-of-features approach. In *Proceedings of the 2nd ACM SIGHIT international health informatics symposium*, pages 631–640, 2012.