Westfälische Wilhelms-Universität
Münster
Fachbereich Mathematik und Informatik

Bachelorarbeit

Erkennung und Klassifikation der Bewegungsmuster von Fußgängern

Erstgutachter: Prof. Dr. Jan Vahrenhold

Zweitgutachter: Dipl.-Inf. Sylvie Temme

Vorgelegt von: Michael Beckemeyer
Steinbreede 12
49497 Mettingen

Matrikelnummer: 382 861

18. Februar 2015

Plagiatserklärung

Hiermit versichere ich, dass die vorliegende Arbeit über *Erkennung und Klassifikation der Bewegungsmuster von Fußgängern* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

(Ort, Datum) (Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1. Einleitung

Durch moderne Technologie können Bewegungsdaten von Personen schon mit einfachen Mitteln erfasst werden. Verbreitete *Smartphones* verfügen über die Fähigkeit, die Position des Benutzers präzise zu bestimmen. Ein wesentlicher Aspekt bei der Bewegung von Menschen ist die Interaktion miteinander. Indem die Bewegungsdaten von Personen aufgezeichnet und miteinander in Relation gestellt werden, können Aussagen über vorliegende Bewegungsmuster getroffen werden.

Diese Arbeit untersucht ein Verfahren, um aus den Bewegungsdaten von Fußgängern auf Bewegungsmuster zu schließen, die mit dem Gruppenverhalten von Menschen zusammenhängen. Besonders relevant bei der Betrachtung von Gruppen ist das Anführer- bzw. Nachfolgerverhalten einzelner Personenpaare. Bei zwei Personen, die sich gemeinsam fortbewegen, lässt sich eine Ähnlichkeit der Bewegungen feststellen. Ausgehend von einer Reihe paarweiser Beziehungen lassen sich Gruppen als Graphen konstruieren, deren Mitglieder sich jeweils gemeinsam bewegen. Mithilfe von Graphenanalyse können schließlich die Anführer dieser Gruppen erkannt werden.

Wesentliche Grundlage der Arbeit ist die Forschung von Kjærgaard und Blunck auf diesem Gebiet. Sie beschreiben einen Ansatz, um die Bewegungsmuster von Passanten bezüglich mehrerer Kriterien zu klassifizieren und geben mehrere Methoden an, mit denen sich die Ähnlichkeit der Bewegungsmuster feststellen lässt. Es kann erkannt werden, ob sich zwei Personenpaare gemeinsam fortbewegen und, falls dies der Fall ist, ob die Personen eine leitende oder folgende Rolle annehmen [KB14].

Auf dieser Forschung aufbauend wird in dieser Arbeit eine leistungsfähige Implementierung des Verfahrens angefertigt. Die dafür nötigen algorithmischen Vorausetzungen werden detailliert beschrieben. Außerdem wird eine weitere Methode zur Messung der Ähnlichkeit von Bewegungsmustern ergänzt. Sowohl die Qualität der Aussagen über die Bewegungsmuster als auch die Leistungsfähigkeit der Implementation werden anhand mehrerer Datensätze bewertet.

Aufbau dieser Arbeit

In Kapitel 2 wird ein Überblick über das Verfahren gegeben. Für die Arbeit wichtige Konzepte werden genannt und definiert. Kapitel 3 enthält die detaillierte Beschreibung der wichtigsten algorithmischen Bausteine, die für die Umsetzung benötigt werden. Der von Kjærgaard und Blunck [KB14] entwickelte Algorithmus wird in Kapitel 4 diskutiert.

Kapitel 5 enthält einen Überblick über die für die Arbeit angefertigte Implementierung des Verfahrens. In Kapitel 6 werden die durchgeführten Experimente beschrieben und die daraus gewonnenen Ergebnisse präsentiert. Eine Zusammenfassung der Arbeit befindet sich in Kapitel 7.

Der Programmtext sowie die verwendeten Datensätze befinden sich auf einem Datenträger im Anhang B. Dies schließt die in vollautomatischen Skripten umgesetzten Experimente ein.

2. Begrifflichkeiten

Bewegungsmuster

Der in dieser Arbeit implementierte Algorithmus hat das Ziel, aus den Bewegungsdaten einer Anzahl von Personen auf verschiedene Muster in ihrem Verhalten zu schließen. *Bewegungsmuster* im Allgemeinen sind zu beobachtende räumliche oder zeitliche Regelmäßigkeiten in den Trajektorien von betrachteten Subjekten oder andere, anwendungsspezifische Beziehungen [DWL08]. Diese Arbeit beschränkt sich auf die Betrachtung von zwei Bewegungsmustern, die mit dem Gruppenverhalten von Fußgängern in Verbindung stehen: das Nachfolgeverhalten von Personen (*following patterns*) und das Anführen von Gruppen (*leadership patterns*). Ein *following pattern* ist eine Beziehung zwischen zwei Personen, bei der die Bewegungen der eine Person die der anderen über einen gewissen Zeitraum vorschreiben [KB14]. Für ein *leadership pattern* ist das Vorhandensein mehrerer Personen (einer Gruppe) Voraussetzung. Es tritt dann auf, wenn eine Person für einen bestimmten Zeitraum als Anführer der Gruppenmitglieder auftritt, aber selbst keiner anderen Person folgt [DWL08].

Anführer und Nachfolger

Beide genannten Bewegungsmuster implizieren, dass sich die beteiligten Personen für den Zeitraum, über den das Muster auftritt, gemeinsam bewegen. Das heißt, dass für diesen Zeitraum eine Ähnlichkeit ihrer Bewegungsdaten vorliegt. Eine solche Ähnlichkeit kann aus einer geringen räumlichen Distanz der Personen folgen, ist aber nicht darauf beschränkt. Ist eine gewisse Ähnlichkeit gegeben, so gilt das Paar als *co-moving*. Umgekehrt ist also eine Klassifizierung als *co-moving* ein erster Schritt, um Anführer- und Nachfolgerbeziehungen zu erkennen. Ist ein Personenpaar A, B *co-moving*, so gibt es für diese Arbeit drei relavante Möglichkeiten: Person A führt B, Person B führt A oder, der

Randfall, sie gehen nebeneinander. A und B heißen in diesen Fällen auch *Anführer* (*leading*) und *Nachfolger* (*following*), im dritten Fall werden sie beide als Anführer aufgefasst (*co-leading*).

Gruppenanführer

Indem die Beziehungen mehrerer Personenpaare kombiniert werden, können *Gruppen* erkannt werden. Zu diesem Zweck wird die Relation *co-moving* als transistiv interpretiert: Wenn sich die Personenpaare (A, B) und (B, C) gemeinsam fortbewegen, so sind auch A und C *co-moving*. Die Menge aller Personen, die paarweise als *co-moving* erkannt wurden, ist in diesem Kontext eine Gruppe. Auf Grundlage der Anführer- und Nachfolgerbeziehungen kann innerhalb einer Gruppe darauf geschlossen werden, für welche Person ein *leadership pattern* vorliegt. Dazu wird auch die Anführer- und Nachfolgerbeziehung als transistiv definiert, sodass der Anführer einer Gruppe intuitiv derjenige ist, dem alle anderen Gruppenmitglieder – auch indirekt – folgen.

Signalstärken und Positionsdaten

Um die Bewegungsmuster von Personen zu analysieren, werden präzise Messungen über einen gewissen Zeitraum benötigt. Die geforderte Präzision folgt aus der Tatsache, dass schon ein Unterschied von wenigen Zentimetern die Einordnung von Personen eines Paares als Anführer oder Nachfolger beeinflussen kann.

Ein Verfahren auf dem Stand der Technik, um präzise Bewegungsdaten zu erhalten, ist die Messung von Signalstärken naheliegender WLAN Access Points mit einem Smartphone. Mithilfe von *Radio Location Fingerprinting* können aus diesen Signalstärken räumliche Positionen bestimmt werden [Kjæ10]. Indem die Positionen jeder betrachteten Person über einen Zeitraum gemessen werden, erhält man deren Trajektorien. Sowohl diese Trajektorien wie auch die rohen Signalstärkedaten werden hier benutzt, um Bewegungsmuster zu erkennen.

Merkmalsvektoren

Ein Merkmalsvektor (*feature vector*) ist ein Vektor, der die numerischen Merkmale eines Objekts enthält. Der Begriff stammt aus dem Gebiet des maschinellen Lernens (*machine*

learning). In dieser Arbeit speichert ein Merkmalsvektor die Ähnlichkeitsmerkmale zu einem Personenpaar und einem Zeitpunkt.

Übersicht über den Algorithmus

Es folgt eine Übersicht über den von Kjærgaard und Blunck entwickelten Algorithmus [KB14], um das Zusammenspiel der oben genannten Begriffe zu verdeutlichen.

1. Gewinnung von Merkmalsdaten

Unter der Verwendung verschiedener Verfahren (präsentiert in Sektion 4.1) wird die *Ähnlichkeit* (*similarity*) der Bewegungsdaten von Personenpaaren bewertet. Um nicht nur eine Aussage über räumliche, sondern auch zeitliche Beziehungen zwischen den Bewegungsdaten der Personen zu erhalten, werden zeitlich verzögerte Datenfolgen (*time-lagged sequences*) eingesetzt. Ein Merkmalsvektor speichert zu mehreren betrachteten Zeitverzögerungen jeweils einen Ähnlichkeitswert.

Ein besonders wichtiges Verfahren zur Bestimmung der Ähnlichkeit der Bewegungsdaten ist *Dynamic Time Warping*. Der Algorithmus wird in Sektion 3.1 detailliert beschrieben.

2. Erkennung sich gemeinsam bewegender Personen

Aus den Merkmalsdaten von **1.** kann mithilfe von maschinellem Lernen darauf geschlossen werden, ob sich zwei Personen zu einem Zeitpunkt gemeinsam bewegen. Abhänging von der Ähnlichkeit der Bewegungen eines Paares findet unter Berücksichtigung des zeitlichen Aspekts eine Klassifizierung als *co-moving* statt.

3. Erkennung von *following patterns*

Nachdem ein Paar als *co-moving* eingestuft wurde, können Nachfolger und Anführer erkannt werden. Dies geschieht durch die Benutzung der zeitlichen Aspekte der Ähnlichkeitswerte in den Merkmalsvektoren. Folgt eine Person einer anderen, so wird erwartet, dass die Bewegungen sich bei einer zeitlichen Verzögerung ähneln, d.h. die Bewegungsdaten der einen Person schreiben die der anderen vor.

Aus den Merkmalsdaten wird eine zeitliche Verzögerung geschätzt, bei der sich die Bewegungsdaten besonders stark ähneln. Von diesem Wert ausgehend wird eine Person als Anführer oder Nachfolger klassifiziert. Falls diese Verzögerung allerdings nicht über einen gewissen Wert hinausgeht, wird das Paar als *co-leading* klassifiziert.

4. Bestimmung der Gruppenanführer

Aus den Beziehungen in **3.** lässt sich ein gerichteter Graph konstruieren. Zusammenhängende Abschnitte dieses Graphen werden als Gruppen interpretiert. Dies sind Personenmengen, die einander direkt oder indirekt folgen. Der *PageRank*-Algorithmus wird verwendet, um zu jeder Gruppe einen Anführer zu finden (Beschreibung in Sektion 3.2).

Die detaillierte Spezifizierung der hier genannten Schritte befindet sich in Kapitel 4.

3. Algorithmische Bausteine

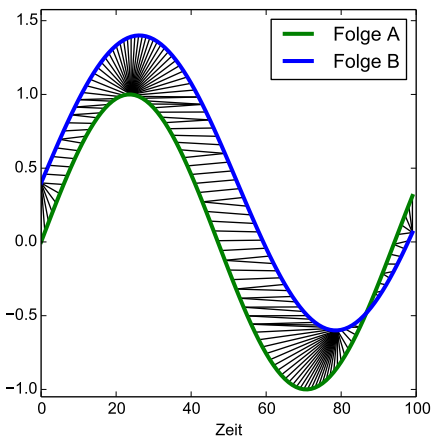
In diesem Kapitel werden zwei für diese Arbeit zentrale algorithmische Bausteine beschrieben: *Dynamic Time Warping* und *PageRank*.

3.1. Dynamic Time Warping

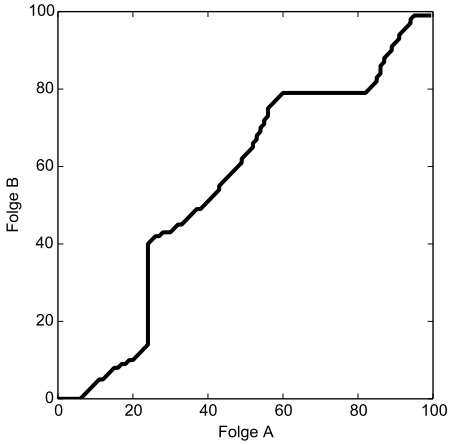
Dynamic Time Warping (im Folgenden auch *DTW*) ist ein Algorithmus, der durch nichtlineares Verzerren (engl. *warping*) zwei Datenfolgen einander angleicht. Aus dem Aufwand, der für die Verzerrung nötig ist, ergibt sich ein Maß für die Ähnlichkeit der beiden Datenfolgen. Der besondere Vorteil von DTW ist, dass der Algorithmus im Vergleich zu gewöhnlichen Distanzfunktionen, wie z.B. der euklidischen Metrik, sowohl mit verzerrten wie auch verschieden langen Datenfolgen umgehen kann.

Der Algorithmus wurde ursprünglich von Sakoe und Chiba entwickelt, um im Feld der Spracherkennung angewendet zu werden[SC78]. Ziel von DTW war es, Aufnahmen von Sprache auf Gleichheit zu überprüfen. Hier ist es möglich, dass eine Aufnahme im Vergleich zu einer gespeicherten Referenzdatenfolge über eine andere Dauer verfügt (etwa durch schnelle Aussprache) oder an einigen Stellen verzerrt ist (z.B. durch deutliche Aussprache von Vokalen). Trotzdem können beide Datenfolgen den gleichen sprachlichen Inhalt repräsentieren.

Inzwischen wird DTW in einer Vielzahl von Gebieten eingesetzt, unter anderem Data Mining [RCM+12] und Bilderkennung [BZ04]. Im Allgemeinen wird DTW häufig auf Zeitreihen von Messwerten angewandt, ist aber nicht auf diese beschränkt – es können auch vollkommen andere Dimensionen verzerrt werden. Beispielsweise kann die Ähnlichkeit von zwei Zeichenketten zu bestimmt werden, indem eine Funktion angegeben wird, die die Distanz zwischen zwei einzelnen Zeichen liefert.



(a) Zwei ähnliche Folgen.



(b) Der resultierende *warping path*.

Abbildung 3.1.: Eine Anwendung des DTW-Algorithmus. Elemente der Folgen werden aufeinander abgebildet.

Es existieren viele Variationen von Dynamic Time Warping. Die folgende Beschreibung orientiert sich stark an dem von Müller beschriebenen Algorithmus [Mü07], der auch in der verbreiteten Python-Bibliothek *mlpy* implementiert ist [AVM+12].

Definitionen

Seien A und B zwei Folgen, deren Werte Elemente einer Menge von Messwerten M sind:

$$A = a_1, a_2, ..., a_n \qquad a_i \in M, n > 0, \tag{3.1}$$

$$B = b_1, b_2, ..., b_m \qquad b_j \in M, m > 0. \tag{3.2}$$

Häufig ist $M = \mathbb{R}^k$, z.B. mit $k = 3$ für Folgen dreidimensionaler Messwerte. In Abbildung 3.1 ist $d = 1$. Im Folgenden werden i und j immer als gültige Indizes für die Folgen A bzw. B verwendet. m und n bezeichnen ausschließlich deren Länge.

Um Elemente beider Folgen vergleichen zu können, wird eine *lokale Kostenfunktion* benötigt:

$$d : M \times M \to \mathbb{R}_{\geq 0}. \tag{3.3}$$

d liefert ein Maß für die Ähnlichkeit von zwei Werten. Anschaulich gibt die Funktion die Kosten für die Verzerrung eines Wertes x auf den Wert y an. Was „Ähnlichkeit" bedeutet, ist von der jeweiligen Anwendung abhängig. Für gewöhnlich wird die Manhattandistanz

oder die euklidische Metrik gewählt, sodass d zusätzlich die Kriterien einer Metrik erfüllt. Streng genommen ist dies aber nicht nötig: Es genügt wenn $d(x, y)$ für ähnliche Werte einen geringen und für sehr verschiedene einen großen Betrag annimmt. Ziel des Algorithmus ist es nun, eine Abbildung der Elemente von A auf B zu finden, sodass die *Kosten* dieser Abbildung in Bezug auf d minimal sind. Eine solche Abbildung heißt *warping path*.

Definition 1. Ein *warping path* $w = w_1, ..., w_p$ (im Folgenden auch Warp-Pfad oder nur Pfad) ist eine Folge von Indizes von A und B, d.h. $w_k = (i_k, j_k)$ mit $0 \leq i_k \leq n$ und $0 \leq j_k \leq m$. Des Weiteren erfüllt er die folgenden drei Eigenschaften:

1. Randbedingung: $w_1 = (1, 1)$, $w_p = (n, m)$.
2. Monotonie: Für $k > 1$ gilt $i_k \geq i_{k-1}$ und $j_k \geq j_{k-1}$.
3. Kontinuität: Für $k > 1$ gilt $i_k - i_{k-1} \leq 1$, $j_k - j_{k-1} \leq 1$ und $w_k \neq w_{k-1}$. [1]

Ein *warping path* ist also eine besondere Abbildung der Elemente von A auf die Elemente von B. Gilt $w_k = (i, j)$, so wird an Stelle k der Wert a_i auf den Wert b_j *gewarpt*. Die drei Eigenschaften bedeuten zusammengenommen, dass ein Pfad jeden Index aus A und B mindestens einmal enthält (folgt sofort aus Monotonie und Randbedingung). Gleichzeitig darf ein Pfad aufgrund der Kontinuität und Monotonie sich in einem Schritt niemals mehr als einen Index in „i-Richtung" und in „j-Richtung" fortbewegen. Es ergibt sich somit für die Elemente eines Pfades w der Länge p die folgende Aussage:

$$1 < k < p : w_k = \begin{cases} (i_{k-1} + 1 & , j_{k-1}) \\ (i_{k-1} & , j_{k-1} + 1) \\ (i_{k-1} + 1 & , j_{k-1} + 1). \end{cases} \tag{3.4}$$

Ein kürzester *warping path* ist der Pfad, der so oft wie möglich das dritte *Schrittmuster* wählt, d.h. gleichzeitig in i- wie auch in j-Richtung weiterführt. Ein solcher Pfad ist z.B. $w = (1, 1), (2, 2), ..., (n, m)$ mit Länge $p = \max(n, m)$. Abhängig von den Längen wiederholt sich entweder n oder m am Ende des Pfades. Die längsten Pfade sind diejenigen, die

[1] Die Bedingung der Ungleichheit findet sich nicht in der Literatur. Hier wird sie nur angegeben, um Verwirrung zu vermeiden. In einem Warp-Pfad machen sich wiederholende Einträge in keinem Fall einen Sinn.

sich zuerst nur in i-Richtung und anschließend in j-Richtung bewegen (oder umgekehrt). Insgesamt gilt also für die Länge p eines Pfades:

$$\max(n, m) \leq p \leq n + m. \tag{3.5}$$

Definition 2. Die mit einem Warp-Pfad w assoziierten *Kosten* $D(w)$ ergeben sich aus der Summe der Kosten der einzelnen Elementabbildungen, d.h. in Bezug auf die in Formel (3.3) genannte Kostenfunktion d:

$$D(w) := D(w_1, ..., w_p) := \sum_{k=1}^{p} d(a_{i_k}, b_{j_k}). \tag{3.6}$$

Definition 3. Ein *optimaler warping path* w_{opt} ist ein Warp-Pfad, der von allen Pfaden über die geringsten Kosten verfügt:

$$\begin{aligned} W &:= \{w \mid w \text{ ist warping path}\}, \\ w_{opt} &:= \operatorname*{argmin}_{w \in W} D(w). \end{aligned} \tag{3.7}$$

Die Distanz (in Bezug auf Dynamic Time Warping) der beiden Folgen A und B ist definiert als die Kosten des optimalen Pfades:

$$DTW(A, B) := D(w_{opt}) = \min\{D(w) \mid w \text{ ist warping path}\}. \tag{3.8}$$

DTW_{norm} ist die in Bezug zur Zeit normalisierte Distanz (*time normalized distance*):

$$DTW_{norm}(A, B) := \frac{1}{n + m} DTW(A, B). \tag{3.9}$$

Im Allgemeinen existiert kein eindeutiger optimaler *warping path*, da nicht auszuschließen ist, dass mehrere Pfade w denselben Wert $D(w)$ annehmen. Alle optimalen Pfade haben aber per Definition dieselben Kosten, sodass $DTW(A, B)$ trotzdem wohldefiniert ist. Für die Berechnung würde es genügen, einen dieser optimalen Pfade zu finden.

Lemma 1. DTW ist genau dann symmetrisch, wenn d symmetrisch ist.

Beweis. Sei w_{opt} ein optimaler *warping path* von A zu B. Dann erhält man einen Warp-Pfad w'_{opt} zwischen B und A, indem alle Indizes in w_{opt} vertauscht werden. Wegen For-

mel (3.6) gilt $D(w_{opt}) = D(w'_{opt})$ genau dann, wenn d symmetrisch ist. w'_{opt} ist ein optimaler Pfad von B nach A: Gäbe es einen günstigeren Pfad von B nach A, so würde dieser (mit umgekehrten Indizes) ein genauso günster Pfad von A nach B sein und w_{opt} wäre nicht optimal ↯. Es gilt also $DTW(A, B) = DTW(B, A)$. □

Abbildung 3.1 auf Seite 11 zeigt in Darstellung (a) zwei aus jeweils 100 Messwerten gewonnene Folgen, die ähnlich verlaufen, aber nicht identisch sind. Darstellung (b) enthält einen optimalen *warping path* zwischen den beiden Folgen. Als lokale Kostenfunktion d diente der Betrag der Differenz von zwei Werten. Verläuft der Pfad durch den Punkt (i, j), so wird in (a) der Messwert an Stelle i in Folge A auf den Wert an Stelle j in Folge B abgebildet (auch durch schwarze Linien in (a) visualisiert).
Gut erkennbar sind die Stellen, an denen ein Wert der einen Folge auf mehrere Werte der anderen abgebildet wird (horizontale bzw. vertikale Linien in (b)). Diagonale Abschnitte signalisieren, dass eine 1:1-Zuordnung stattfindet. Würden die Kurven gleich verlaufen, so wäre der Pfad eine Diagonale.

Berechnung von DTW

Algorithmus 1 zeigt eine naive und rekursive Umsetzung der oben definierten Funktion $DTW(A, B)$. d ist die lokale Kostenfunktion. Für eine Folge $X = x_1, x_2, ..., x_k$ ist $X[1 : l]$ die Teilfolge von X, die die ersten l Elemente enthält, d.h. $X[1 : l] = x_1, ..., x_l$ mit $l \leq k$.

Satz 1. Der Algorithmus DTW_REC berechnet für zwei Folgen A, B auf implizite Art und Weise einen optimalen *warping path* w_{opt} und liefert dessen Kosten. Es gilt mit D aus Definition 3:

$$\text{DTW\_REC}(A, B) = DTW(A, B) = D(w_{opt}). \tag{3.10}$$

Beweis [nach Mü07]. Der Algorithmus terminiert für jede Eingabe. In jedem Aufruf von DTW_REC verringert sich die Länge von mindestens einem der beiden Parameter um 1. Die Rekursion wird also zwangsläufig nach einer endlichen Anzahl von Aufrufen den Basisfall $n = 1$ und $m = 1$ erreichen.

Für die Korrektheit müssen vier Fälle unterschieden werden:

Algorithmus 1 Naive Rekursion zur Berechnung von DTW

```
function DTW_REC(A, B)
    n ← length of A
    m ← length of B
    c ← d(A[n], B[m])
    if n = 1 and m = 1 then                                   ▷ Fall 1
        return c
    else if m = 1 then                                        ▷ Fall 2
        return c + DTW_REC(A[1 : n − 1], B)
    else if n = 1 then                                        ▷ Fall 3
        return c + DTW_REC(A, B[1 : m − 1])
    else                                                      ▷ Fall 4
        c1 ← DTW_REC(A[1 : n − 1], B[1 : m])
        c2 ← DTW_REC(A[1 : n], B[1 : m − 1])
        c3 ← DTW_REC(A[1 : n − 1], B[1 : m − 1])
        return c + min(c1, c2, c3)
    end if
end function
```

1. Beide Folgen haben die Länge 1. Es existiert nur ein Warp-Pfad zwischen den beiden Folgen: $w := (1, 1)$. Dieser ist also optimal und die mit ihm assoziierten Kosten sind $D(w) = D(w_{opt}) = d(a_1, b_1)$ (dies ist c im Pseudocode).

2. Die Länge von B ist 1. Es existiert nur der Pfad $w := (1, 1), (2, 1), ..., (n, 1)$ und dieser ist optimal. Es gilt $D(w) = D(w_{opt}) = \sum_{i=1}^{n} d(a_i, b_1)$. Dieser Wert wird von Bedingung 2 rekursiv berechnet.

3. Die Länge von A ist 1. Analog zu Fall 2: $w := (1, 1), (1, 2), ..., (1, m)$ und $D(w) = D(w_{opt}) = \sum_{j=1}^{m} d(a_1, b_j)$ (berechnet von Bedingung 3).

4. Alle anderen Längen n, m von A bzw. B. Der Beweis erfolgt durch vollständige Induktion über n und m, mit den in Fall 1 bis 3 aufgeführten Basisfällen. Die Induktionsannahme ist die Behauptung. Laut Definition endet jeder *warping path* zwischen A und B mit (n, m). Das vorherige Element im optimalen Pfad w_{opt} ist mit Formel (3.4) entweder $(n - 1, m), (n, m - 1)$ oder $(n - 1, m - 1)$. Seien c_1, c_2, c_3 die Ergebnisse von DTW_REC angewandt auf die drei möglichen Parameterpaare

$$c_1 := \text{DTW\_REC}(A[1 : n - 1], B)$$
$$c_2 := \text{DTW\_REC}(A, B[1 : m - 1])$$
$$c_3 := \text{DTW\_REC}(A[1 : n - 1], B[1 : m - 1])$$

und w_1, w_2, w_3 die dazugehörigen Pfade, sodass $c_1 = D(w_1), c_2 = D(w_2), c_3 = D(w_3)$.
Induktionsschritt: w_1, w_2, w_3 sind für ihre jeweiligen Teilfolgen optimale Pfade, d.h. sie sind die kostengünstigsten Pfade, die in $(1,1)$ beginnen und in $(n-1,m)$ bzw. $(n,m-1)$ bzw. $(n-1,m-1)$ enden. Sei jetzt $w' = w'_1, ..., w'_p$ der *warping path* mit den geringsten Kosten, also $w' := \text{argmin}_{w^* \in \{w_1,w_2,w_3\}} D(w^*)$. Außerdem sei $w := w'_1, ..., w'_p, (n,m)$. Es folgt, dass w ein optimaler *warping path* zwischen A und B ist und es gilt $D(w) = D(w_{opt}) = D(w') + d(a_n, b_m)$. Dies ist das Ergebnis von Bedingung 4 des Algorithmus.

□

Obwohl der oben angegebene Algorithmus das korrekte Ergebnis liefert, ist er ineffizient. Zwar wird das Problem *DTW* in viele einzelne Teilprobleme zerlegt (die rekursiven Aufrufe), diese überschneiden sich jedoch häufig, sodass sie wiederholt berechnet werden. Mithilfe der Technik der *dynamischen Programmierung* lässt sich der Algorithmus verbessern. Dazu ist die Beobachtung erforderlich, dass es nur nm verschiedene zu berechnende Teilprobleme gibt, obwohl die Rekursion in DTW_REC exponentieller Natur ist: für jedes Paar von Teilfolgen der Form $(A[1:i], B[1:j])$ mit $1 \leq i \leq n, 1 \leq j \leq m$ gibt es genau ein Ergebnis. Algorithmus 2 macht sich dies zu Nutze, indem er die Teilergebnisse in einer *Kostenmatrix* C speichert.

Satz 2. Der Algorithmus DTW_DYN berechnet für jede Eingabe dasselbe Ergebnis wie DTW_REC. Es gilt also für zwei Folgen A und B: $\text{DTW\_DYN}(A,B) = DTW(A,B)$.

Beweis. Ziel des Algorithmus ist, die Kostenmatrix C so zu füllen, dass für zwei Indizes i, j die Zelle $C[i,j]$ das Ergebnis des Teilproblems $\text{DTW\_REC}(A[1:i], B[1:j])$ enthält. Im Pseudocode wurden die Stellen gekennzeichnet, die den vier Fällen aus Algorithmus 1 entsprechen. Für zwei Folgen der Länge 1 liegt mit $d(A[1], B[1])$ dasselbe Ergebnis in $C[1,1]$ (Fall 1). Für die erste Spalte der Matrix (also $j = 1$) gilt offensichtlich nach Ausführung der Schleife in Fall 2 für alle $2 \leq i \leq n : C[i,1] = \sum_{k=1}^{i} d(A[k], B[1])$. Dies ist das Resultat von $\text{DTW\_REC}(A[1:i], B[1:1])$. Analoges gilt für die erste Zeile ($i = 1$): hier ist nach Fall 3 für alle $2 \leq j \leq m$ die Gleichung $C[1,j] = \sum_{k=1}^{j} d(A[1], B[k])$ erfüllt. Alle anderen $C[i,j]$, d.h. für $2 \leq i \leq n$ *und* $2 \leq j \leq m$, werden von der Schleife in Fall 4 berechnet. Die Formulierung der Berechnung von $C[i,j]$ ist das exakte Äquivalent zur Rekursion in Fall 4 von Algorithmus DTW_REC. Da in jedem Fall $C[n,m]$ das Ergebnis der Funktion ist, folgt die Behauptung. □

Algorithmus 2 DTW nach Anwendung des Prinzips der dynamischen Programmierung

```
function DTW_DYN(A, B)
    n ← length of A
    m ← length of B
    C ← matrix(n, m)
    C[1, 1] ← d(A[1], B[1])                                  ▷ Fall 1
    for i ← 2, ..., n do                                     ▷ Fall 2
        C[i, 1] ← C[i − 1, 1] + d(A[i], B[1])
    end for
    for j ← 2, ..., m do                                     ▷ Fall 3
        C[1, j] ← C[1, j − 1] + d(A[1], B[j])
    end for
    for i ← 2, ..., n do                                     ▷ Fall 4
        for j ← 2, ..., m do
            c ← d(A[i], B[j])
            C[i, j] ← min(C[i − 1, j], C[i, j − 1], C[i − 1, j − 1]) + c
        end for
    end for
    return C[n, m]
end function
```

Laufzeit: Die Funktion DTW_DYN berechnet eine Matrix C der Dimension $n \times m$. Die Funktion d ist von den Eingangsgrößen A und B unabhängig und benötigt daher nur eine konstante Anzahl von Rechenoperationen. Die Berechnung eines jeden Elements von C erfolgt also in konstanter Zeit (Schleifenkörper von Fall 2, 3, 4). Jedes Element wird nur einmal berechnet. Es folgt, dass sowohl asymptotische Laufzeit als auch Speicherbedarf in $\mathcal{O}(nm)$ liegen.

Berechung des Warp-Pfades

Der Algorithmus DTW_REC berechnet nicht den optimalen Warp-Pfad w_{opt}, sondern nur dessen Kosten $D(w_{opt})$. Allerdings geschieht die Berechnung implizit immer genau dann, wenn sich der Algorithmus für eine „Richtung" entscheiden muss (als Fälle 2 bis 4 markiert). Es wäre also leicht, DTW_REC so zu erweitern, dass der Pfad zurückgegeben wird. Dieser Ansatz würde aber die gleichen Probleme erben, die schon DTW_REC unbrauchbar machten. Besser ist es, sich die oben berechnete *Kostenmatrix* C zu nutzen, die in jedem Element $C[i, j]$ die Kosten des optimalen Pfades von $(1, 1)$ bis (i, j) enthält. Im folgenden Algorithmus sind die anfänglichen Werte für n und m die Länge von A bzw. die Länge

von B. „x . y" bezeichnet die Konkatenation von zwei Listen.

Algorithmus 3 Berechnung des optimalen Pfades w_{opt} zwischen zwei Folgen

```
function WARPING_PATH(C, n, m)
    w ← (n, m)
    if n = 1 and m = 1 then
        return w
    else if n = 1 then
        return WARPING_PATH(C, 1, m − 1) . w
    else if m = 1 then
        return WARPING_PATH(C, n − 1, 1) . w
    else
        i, j ← argmin{C[n − 1, m], C[n, m − 1], C[n − 1, m − 1]}
        return WARPING_PATH(C, i, j) . w
    end if
end function
```

Satz 3. WARPING_PATH (Algorithmus 3) liefert für zwei Folgen A und B einen optimalen Warp-Pfad w_{opt}.

Beweis. Die Korrektheit von WARPING_PATH folgt direkt aus dem Beweis zu Algorithmus 1: Hat eine der beiden Folgen die Länge eins, so gibt es nur einen optimalen Pfad. Andererseits gibt es drei Kandidaten für den „Prefix" von w_{opt}. Aus diesen wird mit der Hilfe von C der mit den geringsten Kosten ausgewählt. Da die Matrix C die Kosten aller optimalen Warp-Pfade enthält, ist der Weg der minimalen Kosten gleichzeitig der optimale Warp-Pfad. □

Laufzeit: Alle im Funktionskörper von WARPING_PATH durchgeführten Operationen benötigen nur konstante Zeit, einschließlich des Zugriffs auf die Matrix C, da diese schon vollständig berechnet ist. Die Laufzeit ist also linear in der Anzahl der rekursiven Aufrufe. Die Anzahl dieser Aufrufe ist identisch mit der Länge des Ergebnisses w_{opt}. Nach Formel (3.5) ist diese durch die Summe der Längen der beiden Folgen nach oben beschränkt. Die Laufzeit liegt also in $\mathcal{O}(n + m)$.

Wie schon in Definition 3 erwähnt, ist der optimale *warping path* im Allgemeinen nicht eindeutig. Eine Implementierung sollte daher für die drei möglichen Schritte Prioritäten festlegen. Beispielsweise kann die Diagonale bevorzugt werden, um einen möglichst kurzen Warp-Pfad zu erhalten, falls eine Mehrdeutigkeit vorliegt.

Beispielrechnunug

Seien A und B die beiden Folgen mit den unten definierten Elementen. d ist der Betrag der Differenz:

$$\begin{aligned} A &= (1,3,6) & n &= 3, \\ B &= (1,2,3,4,5) & m &= 5, \\ d(x,y) &= |x-y|. \end{aligned}$$

Intuitiv sind die beiden Folgen sehr ähnlich: würde A auf die Länge 5 verzerrt (mit Interpolation der fehlenden Elemente), so würden die Graphen der beiden Folgen fast identisch verlaufen. Es wird also ein geringer Wert von $DTW(A,B)$ erwartet. Als Zwischenschritt wird der Übersichtlichkeit halber die $n \times m$-Matrix c berechnet, welche alle möglichen lokalen Kosten speichert, also $c_{ij} := d(a_i, b_j)$.

$$c = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 2 & 1 & 0 & 1 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & 1 & 3 & 6 & 10 \\ 2 & 1 & 1 & 2 & 4 \\ 7 & 5 & 4 & 3 & 3 \end{pmatrix}.$$

Die Kostenmatrix C wurde mithilfe der Rechenvorschriften aus Algorithmus 2 gewonnen. Das heißt für das erste Element gilt $C_{0,0} = c_{0,0} = 0$. Für die erste Zeile mit $i = 1, j > 1$ gilt: $C_{1,j} = C_{1,j-1} + c_{1,j}$. Für die erste Spalte mit $j = 1, i > 1$ gilt analog: $C_{i,1} = C_{i-1,1} + c_{i,1}$. Alle anderen Elemente C_{ij} werden berechnet, indem der minimale gültige „Nachbar" (siehe Schrittmuster in Formel (3.4)) mit dem jeweiligen Wert c_{ij} addiert wird. $C_{3,5} = 3$ ist das Ergebnis des DTW-Algorithmus. Der Warp-Pfad zwischen A und B ist $w_{opt} = (1,1),(1,2),(2,3),(2,4),(3,5)$. Für den Übergang von Element 3 zu Element 2 des Pfades gibt es eine Mehrdeutigkeit den minimalen Nachbar betreffend: hier wurde der diagonale Schritt bevorzugt.

Optimierungen

Obwohl der Ansatz mit dynamischer Programmierung im Vergleich zur naiven Lösung eine deutliche Verbesserung ist, kann er mit einer asymptotischen Laufzeit von $\mathcal{O}(nm)$ bzw. $\mathcal{O}(n^2)$ für Folgen gleicher Länge nur für kleine Problemgrößen eingesetzt werden. Es existieren mehrere Strategien, um die Laufzeit zu verbessern:

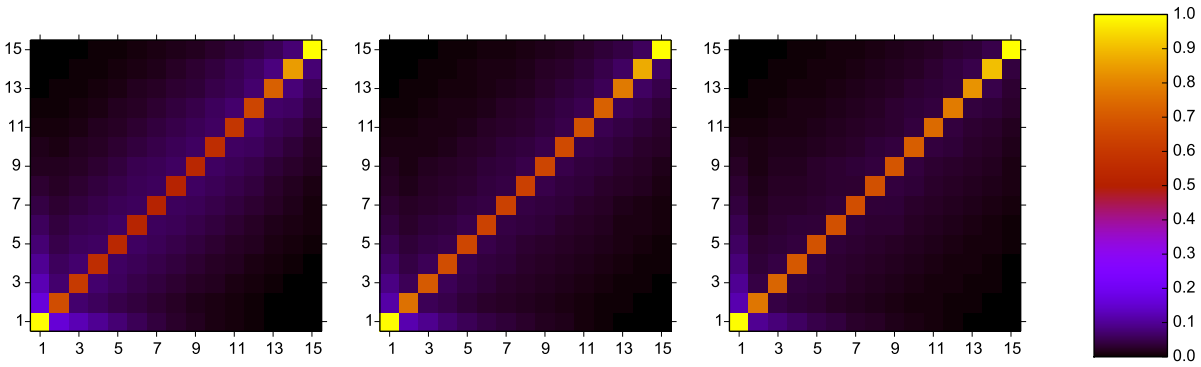


Abbildung 3.2.: Drei Heatmaps, die die Anhäufung von Warp-Pfaden darstellen. Hellere Farben signalisieren eine größere Anhäufung (1.0: Alle Pfade enthalten dieses Indexpaar, 0.0: keine).

Einschränkung des Warp-Pfades

Sakoe und Chiba definieren ein *adjustment window*, um die Anzahl der in der Kostenmatrix C berechneten Einträge zu reduzieren. Dem zugrunde liegt die Beobachtung, dass in vielen Fällen der optimate *warping path* nicht sehr stark von der Diagonale abweicht. Durch die *window length* r wird jeder *warping path* dazu gezwungen, im Band vom Radius r um die Diagonale zu verlaufen. Da es möglich ist, dass der optimale Pfad außerhalb dieses Bandes liegt, gibt der Algorithmus im Allgemeinen nicht das exakte Ergebnis. Obwohl die Laufzeit durch diese Strategie deutlich verbessert wird, liegt sie asymptotisch immer noch in $\mathcal{O}(nm)$ [SC78].

Verringerung der Problemgröße

Indem die Folgen A und B auf zwei neue Folgen von geringer Länge abgebildet werden, kann die Problemgröße reduziert werden. Der DTW Algorithmus wird dann auf diese neuen Folgen angewendet und das Ergebnis als Approxmation des tatsächlichen optimalen Pfades zwischen A und B angesehen. Dieser Ansatz wurde von Keogh und Pazzani gewählt [KP00].

FastDTW

FastDTW wurde von Salvador und Chan entwickelt und reduziert die Auflösung des Problems (wie im vorherigen Punkt) rekursiv. Der resultierende Warp-Pfad einer niedrigen Auflösung dient als als „Suchbereich" für den Warp-Pfad in der höheren Auflösung, bis die volle Auflösung erreicht wurde. Der Algorithmus benötigt für Folgen gleicher Länge nur lineare Zeit [SC07].

SparseDTW

SparseDTW reduziert den Speicherbedarf, indem es die Ähnlichkeit von Folgen ausnutzt. So ist für die Berechnung ähnlicher Abschnitte deutlich weniger Speicher nötig. SparseDTW liefert immer das optimale Resultat. Der Algorithmus wurde von Al-Naymat, Chawla und Taheri entwickelt [ACT12].

Abbildung 3.2 zeigt drei Heatmaps, die aus allen Pfaden von drei verschiedenen Experimenten gewonnen wurden (je eine Darstellung pro Experiment). Für jedes Experiment wurde jeder berechnete *warping path* betrachtet und dessen Indexpaare vermerkt. Jeder Kombination von Indexpaaren wurde eine relative Häufigkeit zugewiesen (1.0: Jeder *warping path* verfügte über dieses Paar, 0.0: keiner). Eine hellere Farbe repräsentiert eine größere Anhäufung. Aus der Abbildung wird deutlich, dass die Pfade sich in einem kleinen Radius um die Diagonale häufen. Ein Band, wie von Sakoe und Chiba beschrieben, würde also gute Resultate zeigen.

3.2. PageRank

PageRank ist ein Verfahren zur Analyse von verlinkten Graphen. Der Algorithmus weist jedem Knoten in einem gerichteten Graphen einen Wert zu, der dessen relative Wichtigkeit repräsentiert. Die ursprüngliche Anwendung von PageRank lag in der Suchmaschine *Google* [BP98]. Dort diente der Algorithmus zur Analyse eines Webgraphen. Ein solcher Graph enthält für jedes Dokument (die *Website*) einen Knoten und für jede Verbindung zwischen diesen eine Kante (der *Hyperlink*). Der PageRank-Algorithmus diente zur Klassifizierung der Websites: Dokumente mit einem höheren Rang wurden bei einer Suche an früherer Stelle genannt. Page, Brin, Motwani und Winograd haben den Algorithmus ausführlich beschrieben [PBMW99].

Anschaulich wird die Wichtigkeit eines Dokuments größer, je öfter auf dieses verwiesen (verlinkt) wird. Dabei wird nicht nur die Anzahl der eingehenden Kanten gezählt, sondern auch deren Quelle in Betracht gezogen. Eine Kante, die von einem Knoten ausgeht, der selbst einen hohen Rang hat, beinflusst den Rang des verlinkten Dokuments beträchtlich. Ein Link ausgehend von einem „unwichtigen" Knoten hat kaum eine Auswirkung.

Der Algorithmus

Definition 4. Sei $G = (V, E)$ ein gerichter Graph und $V = (v_1, ..., v_n)$ dessen Knotenmenge. $E = (e_1, ..., e_m) \subseteq V \times V$ ist die Menge der Kanten von G. Eine gerichtete Kante $e = (v_1, v_2) \in E$ verfügt über die Quelle v_1 und das Ziel v_2. Außerdem sind für $v \in V$ die Menge der Nachfolger und Vorgänger wie folgt definiert:

$$N^+(v) := \{v' \in V \mid (v, v') \in E\}, \qquad N^-(v) := \{v' \in V \mid (v', v) \in E\},$$
$$\deg^+(v) := |N^+(v)|, \qquad \deg^-(v) := |N^-(v)|.$$

Dann lautet die rekursive Definition von PageRank:

$$PR(v) := \frac{1-d}{n} + d \sum_{v' \in N^-(v)} \frac{PR(v')}{\deg^+(v')}. \tag{3.11}$$

$d \in [0, 1]$ wird Dämpfungsfaktor (*damping factor*) genannt. Der von Brin und Page verwendete Wert ist $d = 0.85$ [BP98].

Die Berechnung des Ranges erfolgt über einen iterativen Prozess, bei dem sich der PageRank eines Knotens aus den Ergebnissen seiner Vorgänger aus der letzten Iteration ergibt:

$$PR_{i+1}(v) := \frac{1-d}{n} + d \sum_{v' \in N^-(v)} \frac{PR_i(v')}{\deg^+(v')}. \tag{3.12}$$

Ausgehend von den Startwerten $PR_0(v)$ konvergiert die Folge $PR_i(v)$ gegen $PR(v)$. Für gewöhnlich wird $PR_0(v) = \frac{1}{n}$ gesetzt.

Es an anderer Stelle gezeigt, dass die Folge $PR_i(v)$ sehr schnell konvergiert: für eine Problemgröße von 322 Millionen Kanten wurden beispielsweise nur 52 Iterationen benötigt [PBMW99].

Kantengewichte

Im Algorithmus oben verfügt jede ausgehende Kante eines Knotens v' über die gleiche Gewichtung $\frac{1}{\deg^+(v')}$. Die Gleichung kann so interpretiert werden, dass jeder Knoten seinen Rang gleichmäßig auf alle seine Nachfolger in $N^+(v')$ verteilt. PageRank kann leicht erweitert werden, um eine andere Kantengewichtung zu berücksichtigen. Sei $g : E \to [0, 1]$ eine Funktion, die jeder Kante ein Gewicht zuweist. Außerdem gelte für alle Knoten v mit $\deg^+(v) \neq 0$:

$$1 = \sum_{v' \in N^+} g(v, v'). \tag{3.13}$$

Das heißt, die Summe aller Gewichte von ausgehenden Kanten eines Knotens ist 1. Eine gewichtete Variante von PageRank kann dann wie folgt formuliert werden:

$$PR_g(v) := \frac{1-d}{n} + d \sum_{v' \in N^-(v)} g(v', v) PR(v'). \tag{3.14}$$

Die Berechnung von PR_g erfolgt auf dieselbe Art und Weise wie oben in Formel (3.12), mit Ausnahme der Ersetzung von $\frac{1}{\deg^+(v')}$ durch $g(v', v)$.

4. Algorithmus

4.1. Gewinnung von Merkmalsdaten

Voraussetzungen

Wie in Kapitel 2 erwähnt, werden für den Algorithmus Messungen von Personen über einen gewissen Zeitraum benötigt. In dieser Arbeit werden hauptsächlich Bewegungsdaten betrachtet, die entweder aus Positionsdaten oder den Signalstärken naher WLAN Access-Points in einem Gebäude hervorgehen. Bewegungsdaten aus anderen Quellen sind möglich, sofern die Präzisionsanforderungen erfüllt werden. Im Falle von Positionsdaten wird davon ausgegangen, dass diese zwei- oder dreidimensionalen Koordinaten in einem kartesischen Koordinatensystem entsprechen.[1] Die Signalstärkedaten werden zu jedem in der Nähe befindlichen Access-Point die dazugehörige Signalstärke enthalten. Liegen zu einem gewissen Zeitpunkt k Access Points in der Nähe eines Gerätes, so ist der Messvektor zu diesem Zeitpunkt ein k-dimensionaler Vektor.

Definition 5. Die Anzahl der zu jedem Zeitpunkt zur Verfügung stehenden Messungen ist variabel. Für Signalstärkenmessungen entspricht diese Anzahl den Access Points in der Nähe des Gerätes. Für räumliche Daten ist die Anzahl hingegen immer gleich. Deshalb ist $M := \mathbb{R} \cup \{\varepsilon\}$ die Menge der Messwerte. ε repräsentiert einen fehlenden Wert. In Berechnungen, die auf der Grundlage von räumlichen Daten stattfinden, wird ε nicht vorkommen.

Definition 6. Sei $T := \{1, ..., d\}$ die Menge der Zeitpunkte, zu denen gemessen wurde (d ist die Dauer). Sei $P := \{p_1, ..., p_n\}$ die Menge der n beteiligten Personen (bzw. deren Messgeräte). Die *Messdimension* k ist definiert als die Anzahl der *insgesamt* bekannten

[1] Im Verlauf des Kapitels wird die euklidische Metrik zur Messung von Abständen benutzt. Für ein andersartiges Koordinatensystem muss eine passende Abstandsfunktion gewählt werden.

Dimensionen, d.h. die Anzahl aller im Zeitraum $1, ..., d$ vorkommenden Access Points beziehungsweise die Anzahl der räumlichen Dimensionen.

Sei m die Funktion, die für eine Person und einen Zeitpunkt den *Messvektor* dieser Person liefert:

$$\begin{aligned} m : P \times T &\to M^k, \\ m(p,t) &= (m_1, ..., m_k). \end{aligned} \tag{4.1}$$

Im Messvektor ist m_i der Messwert, der zur Dimension i gehört, d.h. zum entsprechenden Access Point oder zur dazugehörigen räumlichen Dimension.

Definition 7. Kjærgaard und Blunck definieren den Begriff einer zeitlich verzögerten Datenfolge (*time-lagged sequence*) [KB14]. Ein Messvektor zu einem Gerät p und einem Zeitpunkt t, der um einen *time-lag* $l \in \mathbb{Z}$ verzögert wurde, ist definiert als

$$m_l(p,t) := m(p, t+l). \tag{4.2}$$

Offensichtlich liefert ein *time-lag* von 0 die ursprüngliche Sequenz. Anschaulich liefert eine Zeitverzögerung von l die Werte, die l Zeitpunkte in der „Zukunft" liegen (l kann negativ sein, dann sind es Werte aus der „Vergangenheit").

Merkmalsvektoren

Der Algorithmus betrachtet zwei Personen als *co-moving*, wenn die Bewegungen einer Person die einer anderen vorschreiben. Dies ist direkte Folge aus den drei möglichen Beziehungen, die für zwei Personen herrschen können: Person A folgt B, Person B folgt A oder sie gehen nebeneinander. Wenn Person A der Anführer ist, dann wird erwartet, dass die Bewegungsdaten von A die von B „vorhersagen". Das bedeutet, dass die Muster von A, um einen gewissen Zeitraum verzögert, denen von B ähnlich sind. Gilt die umgekehrte Beziehung, dann liegt auch eine Ähnlichkeit der Muster vor, nämlich mit einem umgekehrten Verzögerung. Laufen die beiden Personen nebeneinander, so wird eine große Ähnlichkeit bei einer fehlenden Verzögerung erwartet.

Für die Berechnung der Merkmalsvektoren sind nun zwei Parameter von Bedeutung: Die maximale Verzögerung (*time-lag*) $z \in \mathbb{N}_0$ und die Fenstergröße (*window size*) $w \in \mathbb{N}$. Im vorherigen Paragraphen entspricht z der maximalen Zeitverzögerung, die noch berechnet wird. Die *window-size* bestimmt die Länge des auf Ähnlichkeit überprüften Zeitraums.

Definition 8. Ein *Merkmalsvektor* v (*feature vector*) ist ein Vektor der Länge $2z + 1$, der für jede Zeitverzögerung $l \in \mathbb{Z}$ mit $-z \leq l \leq z$ einen Ähnlichkeitswert enthält. v_l ist der Ähnlichkeitswert zum *time-lag* l. Ein niedriger Betrag dieses Wertes bedeutet große Ähnlichkeit der Bewegungsdaten von zwei Geräten.

Für Geräte p_1, p_2 und einen Zeitpunkt t hat der dazugehörige Merkmalsvektor v die Einträge $v_l = sim(p_1, p_2, t, l, w)$ mit den oben erwähnten Werten von l. Die Funktion *sim* gibt die Ähnlichkeit (*similarity*) der beiden Messfolgen bei einem *time-lag* von l und einer *window size* von w an und wird erst im Folgenden definiert.

Verschiedene Ähnlichkeitsmaße

Ein Maß für die Ähnlichkeit von zwei zeitverzögerten Messfolgen kann auf verschiedene Arten berechnet werden. Seien $a := m_0(p_1, t)$ und $b := m_l(p_2, t)$ die Messvektoren der Geräte p_1 bzw. p_2 zum Zeitpunkt t. Für den Vektor b wird außerdem der *time-lag* l beachtet. Sei I die Menge der Indizes, für die mindestens eines der beiden Geräte einen gültigen (nicht ε) Messwert besitzt:

$$I := \{i \mid \text{falls } a_i \neq \varepsilon \vee b_i \neq \varepsilon\}. \tag{4.3}$$

Für einen Messvektor m sei m^I der Vektor der Dimension $|I|$, der nur noch die Elemente von m enthält, für deren Indizes $i \in I$ gilt.

Die Indexmenge I wird sicherstellen, dass im Fall der Berechnung auf Grundlage von Signalstärken die Anzahl der jeweils betrachteten Werte überschaubar bleibt. Zwar ist es möglich, dass die Gesamtanzahl der bekannten Access Points k sehr groß ist, die Zahl der sich zur einem Zeitpunkt in Reichweite befindenden Access Points kann aber deutlich geringer sein. Deshalb werden für die Ähnlichkeit der Folgen nur die Indizes in I in Betracht gezogen. Für räumliche Koordinaten hat diese Überlegung keine Auswirkung: I ist immer gleich $\{1, ..., k\}$.

Schließlich wird eine Funktion r_l benötigt, welche für eine Person p und einen Zeitpunkt t den *reduzierten Messvektor* liefert. p ist hier entweder p_1 oder p_2. l ist der *time-lag*. Sei $f(m)$ eine Hilfsfunktion, die für einen Messvektor m die Einträge, die den Wert ε haben,

durch einen reellen Wert $S \in \mathbb{R}$ ersetzt (alle anderen Einträge bleiben erhalten).

$$\begin{aligned} r_l : P \times T &\to \mathbb{R}^{|I|}, \\ r_l(p,t) &:= f(m_l(p,t))^I. \end{aligned} \tag{4.4}$$

Für Signalstärkedaten stellt der Parameter S die größtmögliche Distanz von einem Access Point dar, also eine besonders geringe Signalstärke.[2] Mithilfe von S können auch solche Dimensionen betrachtet werden, für die nur eines der Geräte über einen Messwert verfügt.

Jeder der folgenden Abschnitte definiert eine Funktion $sim(p_1, p_2, t, l, w)$, wie sie in Definition 8 benutzt wurde.

Euklidische Distanz. Mithilfe der euklidischen Distanz *ed* wird über einen Zeitraum der Länge w (die *window size*) die Distanz der in der Zeit verschobenen Messvektoren ermittelt. Die Distanzen werden summiert und ihr arithmetisches Mittel dient als Maß der Ähnlichkeit.

$$\begin{gathered} sim_{ed}(p_1, p_2, t, l, w) := \frac{1}{w} \sum_{0 \le j < w} ed(a_j, b_j), \\ \text{mit } a_j = r_0(p_1, t - \lfloor \frac{w}{2} \rfloor + j) \text{ und } b_j = r_l(p_2, t - \lfloor \frac{w}{2} \rfloor + j). \end{gathered} \tag{4.5}$$

Dynamic Time Warping. Dynamic Time Warping (DTW) gibt ein Maß für die Ähnlichkeit von zwei Datenfolgen und wird in Sektion 3.1 ausführlich beschrieben. Betrachtet man einen Vektor $r_l(p,t)$ als einen Zeilenvektor, der für jede Messdimension den dazugehörigen Messwert im Zeitpunkt t enthält, so sind die a und b Spaltenvektoren, in denen die Einträge die Messwerte der gleichen Dimension zu verschiedenen Zeitpunkten sind. Im Folgenden werden die Messwerte jeder einzelnen Messdimension i als eine Folge in der Zeit interpretiert. Die DTW-Ähnlichkeiten der zeitverzögerten Folgen werden aufsummiert und ihr Mittelwert liefert die Ähnlichkeit der Bewegungsdaten. Die *lokale*

[2] In gängigen Anwendungen sind -100 dBm das untere Ende der Messskala für Signalstärken in Bezug auf WLAN.

Kostenfunktion ist der Betrag der Differenz (*Manhattan-Metrik*).

$$
\begin{aligned}
sim_{dtw}(p_1, p_2, t, l, w) &:= \frac{1}{|I|} \sum_{i=1}^{|I|} DTW(a_i, b_i), \\
\text{mit } a_i = \begin{pmatrix} r_0(p_1, t_0)_i \\ \vdots \\ r_0(p_1, t_{w-1})_i \end{pmatrix}, b_i &= \begin{pmatrix} r_l(p_2, t_0)_i \\ \vdots \\ r_l(p_2, t_{w-1})_i \end{pmatrix} \\
\text{und } t_j = t - \lfloor \frac{w}{2} \rfloor + j &\text{ für } 0 \leq j < w.
\end{aligned}
\tag{4.6}
$$

Mehrdimensionales Dynamic Time Warping. Der Ansatz im vorhergehenden Absatz berechnet $|I|$ verschiedene DTW-Werte, wobei für jede Berechnung zwei Folgen von Skalaren benutzt werden. Wie in Sektion 3.1 gezeigt wurde, ist der DTW-Algorithmus nicht auf eindimensionale Folgen beschränkt. In der folgenden Methode gibt es nur zwei Folgen von Messwerten: jeder Eintrag von a oder b ist nun der gesamte Messvektor zu einem bestimmten Zeitpunkt. Sie findet sich nicht in der zugrundeliegenden Arbeit von Kjærgaard und Blunck und wird hier *Multi-DTW* genannt, um sie von der Methode DTW zu unterscheiden. Als lokale Kostenfunktion für DTW dient die euklidische Distanz.

$$
\begin{aligned}
sim_{mdtw}(p_1, p_2, t, l, w) &:= DTW(a, b), \\
\text{mit } a = \begin{pmatrix} r_0(p_1, t_0) \\ \vdots \\ r_0(p_1, t_{w-1}) \end{pmatrix}, b &= \begin{pmatrix} r_l(p_2, t_0) \\ \vdots \\ r_l(p_2, t_{w-1}) \end{pmatrix} \text{ und } t_j \text{ wie in Formel (4.6).}
\end{aligned}
\tag{4.7}
$$

Zusammengefasst ergibt sich für die Berechnung der Merkmalsdaten der Algorithmus 4.

Laufzeit: An dem Algorithmus lässt sich unmittelbar ablesen, dass die Anzahl der ausgeführten Operationen linear ist in den folgenden Variablen: der Länge von *pairs*, d und z (N sei im Folgenden die Anzahl der Paare). Außerdem ist die asymptotische Laufzeit der Funktion *sim* von Belang, welche $\mathcal{O}(N \cdot d \cdot z)$-mal aufgerufen wird. Für sim_{ed} ist die Laufzeit linear in w und aufgrund der eukl. Distanz linear in der Dimension $|I|$ der Messvektoren. $|I|$ ist nach oben durch die Messdimension k beschränkt, es gilt also $sim_{ed} \in \mathcal{O}(w \cdot k)$. Die asymptotische Laufzeit von sim_{dtw} ist zunächst linear in $|I|$. Mit Verweis auf Sektion 3.1 benötigt der DTW-Algorithmus quadratische Zeit, also gilt insgesamt $sim_{dtw} \in \mathcal{O}(w^2 \cdot k)$,

Algorithmus 4 Berechnung der Merkmalsdaten

Input:

pairs Die Paare von Geräten, für die Merkmalsdaten erzeugt werden sollen.

d Die Dauer der Messungen.

z Die maximale zeitliche Verzögerung.

w Die Fenstergröße.

Output:

result Eine Struktur, die für jedes Paar und für jeden Zeitpunkt einen Merkmalsvektor enthalten wird.

```
function FEATURE_DATA(pairs, d, z, w)
    result ← structure(length of pairs × d)      ▷ Genügend Speicher für alle Paare und
                                                   jeweils alle Zeitpunkte.
    for all (p1, p2) in pairs do
        for t ← 1, ..., d do
            v ← vector(2z + 1)                   ▷ Der feature vector. Indizes ab 0.
            for l ← −z, ..., 0, ..., z do
                v[z + l] = sim(p1, p2, t, l, w)  ▷ sim ist eine der Ähnlichkeitsfunktionenen.
            end for
            result[p1, p2][t] ← v
        end for
    end for
    return result
end function
```

da w die Länge aller der Folgen ist. Die *Multi-DTW*-Methode sim_{mdtw} berechnet nur einmal die DTW-Ähnlichkeit – wieder mit Folgenlänge w. Als lokale Kostenfunktion dient nun aber die euklidische Distanz zwischen Vektoren der Dimension $|I| \leq k$, sodass insgesamt auch $sim_{mdtw} \in \mathcal{O}(w^2 \cdot k)$ gilt.

Somit gilt für den gesamten Algorithmus: FEATURE_DATA $\in \mathcal{O}(N \cdot d \cdot z \cdot w^2 \cdot k)$. Unter der Annahme, dass jedes mögliche Gerätepaar betrachtet wird, also $N \in \mathcal{O}(n^2)$, folgt: FEATURE_DATA $\in \mathcal{O}(n^2 \cdot d \cdot z \cdot w^2 \cdot k)$. Für die euklidische Variante ist jeweils w^2 durch w zu ersetzen.

Als mögliche Optimierung kann die Anzahl der betrachteten Paare reduziert werden: Wegen der Symmetrie von DTW und der euklidischen Distanz kann ein Paar (j, i) ausgelassen werden, sofern (i, j) schon berechnet wurde. Dies beeinflusst aber nicht die asymptotische Laufzeit.

4.2. Erkennung sich gemeinsam bewegender Personen

Kjærgaard und Blunck benutzen maschinelles Lernen, um sich gemeinsam bewegende (*co-moving*) Personen zu erkennen. Eine *support vector machine* (SVM) wird mit existierenden Merkmalsvektoren trainiert. Eine *ground truth* gibt beim Training darüber Aufschluss, ob ein Vektor zu einem Gerätepaar gehört, welches sich zu einem bestimmten Zeitpunkt gemeinsam bewegt. Wurde eine SVM mit diesen Daten trainiert, kann sie als Klassifizierer für weiter Merkmalsvektoren benutzt werden. Ein Vektor, der zu einem Gerätepaar und einem Zeitpunkt gehört, wird also von der SVM als *co-moving* oder nicht *co-moving* klassifiziert.

Die genauen Vorgänge beim maschinellen Lernen bieten zwar Gelegenheit zu einer tieferführenden Auseinandersetzung, gehen aber weit über die Zielsetzung dieser Arbeit hinaus.

4.3. Bestimmung der Anführer und Nachfolger

Seien p_1, p_2 ein Gerätepaar und t ein Zeitpunkt. v sei ein dazugehöriger *feature vector*, der durch eine der Methoden aus Sektion 4.1 berechnet wurde. Außerdem sei der Merkmalsvektor von einem Klassifizierer wie in Sektion 4.2 als *co-moving* eingestuft worden. Es gibt dann drei mögliche Situationen, die zwischen p_1 und p_2 herrschen können:

1. p_1 folgt p_2, p_1 ist Nachfolger. Die Bewegungen von p_2 schreiben die von p_1 vor.
2. p_2 folgt p_1, p_1 ist Anführer. Die Bewegungen von p_1 schreiben die von p_2 vor.
3. Sie laufen nebeneinander. Die Bewegungen schreiben sich gegenseitig vor und sind (fast) identisch.

Zu den drei Fällen gilt intuitiv, dass die größte Ähnlichkeit der Bewegungsdaten bei einem 1. negativen *time-lag*, 2. positiven *time-lag* und 3. bei einer Verzögerung, die nah bei 0 liegt, herrschen sollte.

Kjærgaard und Blunck geben zwei verschiedene Möglichkeiten an, um den *time-lag*, bei dem die größte Ähnlichkeit vorliegt, zu schätzen. z ist der maximale *time-lag* aus Sektion 4.1.

$$l_{est} := \underset{-z \leq l \leq z}{\operatorname{argmin}} |v_l|. \tag{4.8}$$

Dieser naive Ansatz wählt strikt den *time-lag*, bei dem v den geringsten Betrag und somit die größte Ähnlichkeit erreicht. Eine Methode, die sowohl in der Arbeit von Kjærgaard und Blunck wie auch in der hier vorgestellten Implementierung etwas bessere Ergebnisse zeigte, lautet wie folgt [KB14]:

$$l_{est} := \frac{1}{s} \sum_{-z \leq l \leq z} \frac{l}{v_l}, \text{ mit } s = \sum_{-z \leq l \leq z} \frac{1}{v_l}. \tag{4.9}$$

Ausgehend von diesem geschätzen *time-lag* kann nun die Beziehunge zwischen p_1 und p_2 erkannt werden:

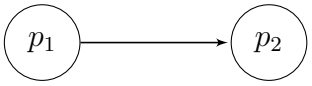
$$rel(p_1, p_2) := \begin{cases} \text{following} & \text{, falls } l_{est} < -th \\ \text{leading} & \text{, falls } l_{est} > th \\ \text{co-leading} & \text{, sonst.} \end{cases} \tag{4.10}$$

$th \in \mathbb{R}_+$ gibt den Bereich um 0 an, in dem ein geschätzer *time-lag* liegen muss, um die Beziehung als *co-leading* einzustufen. Werte, deren Betrag über diese *threshold* hinausgeht, führen zu einer Klassifizierung als *following* oder *leading*. In der Implementierung ist $th = 0.1$ gesetzt.

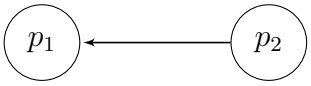
4.4. Bestimmung der Gruppenanführer

Da mit dem Verfahren aus Sektion 4.3 für alle betrachteten Paare die Beziehung *rel* entschieden wird, kann auf Grundlage dieser Informationen ein gerichteter Graph konstruiert werden. Der Graph soll die Anführer- und Nachfolgerbeziehungen zu einem bestimmten Zeitpunkt repräsentieren. Dazu wird zunächst jeder Person ein Knoten zugewiesen. Eine gerichtete Kante zwischen zwei Personen (bzw. deren Knoten) p_1 und $_2$ besagt, dass p_1 p_2 zum betrachteten Zeitpunkt folgt. Abbildung 4.1 visualisiert die vier verschiedenen Zustände, die für Kanten zwischen zwei Personen gelten können. In Abbildung 4.2 ist ein Graph dargestellt, wie er aus solchen Kanten entstehen kann.

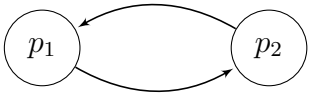
Für diesen Schritt des Algorithmus ist die Problemstellung nun die Identifizierung von Gruppen und deren Anführern. Kjærgaard und Blunck wenden zum Zweck der Gruppenidentifizierung den Begriff der verbundenen Komponenten (*connected components*) aus der Graphentheorie an [KB14]. Ein verbundener (ungerichteter) Graph ist ein Graph, in dem es für jedes Knotenpaar p_1, p_2 einen Weg gibt, der mit p_1 beginnt und mit p_2 endet. Sei $G = (V, E)$ ein gerichter Graph wie z.B. in Abbildung 4.2. Dann wird der ungerichtete



(a) p_1 folgt p_2.



(b) p_2 folgt p_1.



(c) p_1 und p_2 sind *co-leading*.

(d) p_1 und p_2 sind nicht *co-moving*.

Abbildung 4.1.: Die verschiedenen Zustände zwischen Knoten im gerichteten Graphen.

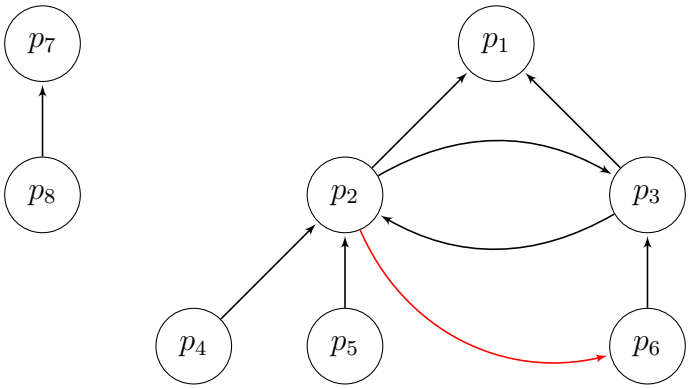


Abbildung 4.2.: Ein Graph, der aus Kanten wie in Abbildung 4.1 entstanden ist.

Graph $G' = (V, E')$ erzeugt, indem jede Kante $(v_1, v_2) \in E$ auf die Kante $\{v_1, v_2\}$ abgebildet wird. Indem nun die Menge der zusammenhängenden Komponenten C von G' bestimmt wird, erhält man gleichzeitig die Menge der Gruppen in G. [3] Mit dieser Definition sind auch Personen eingeschlossen, die sich allein bewegen. Diese werden als Einpersonengruppen aufgefasst. Für das Beispiel in Abbildung 4.2 gilt $C = \{\{p_1, ..., p_6\}, \{p_7, p_8\}\}$, es gibt also zwei Gruppen.

Kjærgaard und Blunck deuten in ihrer Arbeit auf ein Verfahren hin, welches die Knoten eines Graphen bewertet und so den Anführer einer Gruppe findet [KB14, Sektion 4.2]. An dieser Stelle wird der Algorithmus PageRank (siehe Sektion 3.2) genutzt, um eine solche Bewertung durchzuführen. Für jede verbundene Komponente $c = \{v_1, ..., v_n\} \in C$ wird PageRank auf alle Knoten angewandt. Der Knoten mit dem höchsten Rang gilt als der Anführer l_c von c:

$$l_c := \underset{v \in c}{\operatorname{argmax}}(PR(v)). \tag{4.11}$$

Die Benutzung von PageRank hat für diese Anwendung mehrere Vorteile. Zunächst ist die

[3] Die Umwandlung in einen ungerichteten Graphen ist nötig, weil in einem gerichteten Graphen Gruppen, wie sie in der Abbildung gezeigt sind, nicht verbunden wären.

Beobachtung von Bedeutung, dass ein Graph einer Gruppe im Allgemeinen kein Baum ist, in dem zuerst der Anführer steht, dann die Personen, die ihm folgen usw. Beispielsweise dürften Kanten wie die von p_2 zu p_6 oder die zwischen p_2 und p_3 in einem Baum nicht vorkommen. Außerdem kann es in G Zyklen geben (diese sind schon wegen Abbildung 4.1c nicht zu vermeiden). Der Algorithmus PageRank bietet in diesen Fällen die Flexibilität, dass er für jeden gerichteten Graphen anwendbar ist.

In Abbildung 4.2 wurde eine Kante rot markiert. In der Praxis könnte diese Kante durch eine fehlerhafte Bewertung entstanden sein, z.B. in der Klassifizierung als *co-moving* (Sektion 4.2) oder in der Erkennung der „folgen"-Relation (Sektion 4.3). Der Fehler könnte auch auf unzuverlässige Messungen zurückzuführen sein. PageRank kann in solchen Fällen kompensieren, sodass der Gruppenanführer c_l trotzdem noch erkannt werden kann:

- Es können trotz der fehlerhaften Kante genügend korrekte Wege zu c_l führen, sodass dieser genug Rang akkumuliert.
- Die fehlerhafte Kante führt unter Umständen indirekt wieder zu c_l, sodass die Auswirkungen auf das Ranking gering sind.

Schließlich ist die Menge der Anführer zu einem Zeitpunkt definiert als $L := \{l_c \mid c \in G\}$. Für das Beispiel in der Abbildung gilt $L = \{p_1, p_7\}$.

Die Genauigkeit der Methode kann weiter verbessert werden, indem die Kanten des Graphens gewichtet werden. Als Gewicht einer Kante (p_1, p_2) wird hier der Betrag des geschätzen *time-lags* l_{est} zwischen p_1 und p_2 genutzt. Diese Gewichte werden skaliert, sodass die Summe der Gewichte von ausgehenden Kanten 1 ergibt. Die zugrundeliegende Überlegung für diese Wahl ist, dass eine geschätze Verzögerung mit hohem Betrag über eine große Sicherheit verfügt. Hat der Wert allerdings nur einen kleinen Betrag, so ist es nicht unwahrscheinlich, dass ein falsches Vorzeichen gewählt wurde. In diesem Fall würde aufgrund des geringen Gewichtes einer solchen Kante auch deren Relevanz gering sein. Ein Vergleich der beiden Methoden befindet sich in Kapitel 6.

Die asymptotische Laufzeit des Verfahrens hängt ganz von den beiden verwendeten Algorithmen ab: Das Suchen von verbundenen Komponenten findet in linearer Zeit statt [SLL02]. Die Zeitkomplexität von PageRank ist schwer einzuschätzen, der Algorithmus konvergiert allerdings schnell [PBMW99]. In jedem Fall muss zusätzlich die Dauer d des untersuchten Zeitraums einbezogen werden, da für jeden Zeitpunkt die Gruppenanführer bestimmt werden.

Für die Komplexität des Graphen selbst gilt: die Anzahl der Knoten entspricht n, der Anzahl der betrachteten Personen. Die Anzahl der Kanten ist linear in der Anzahl der Paare, liegt also in $\mathcal{O}(n^2)$.

5. Implementation

Die Implementierung der in Kapitel 4 beschriebenen Vorgehensweise ist in zwei Bereiche gegliedert: Eine Bibliothek, welche die verwendeten Algorithmen und Datenstrukturen zur Verfügung stellt sowie mehrere Kommandozeilenprogramme, die diese Bibliothek nutzen und direkt vom Benutzer bedienbar sind. Der Quelltext wurde in der Programmiersprache C++ verfasst. Anhang A gibt eine Übersicht über die Projektstruktur, außerdem befindet sich auf dem Datenträger eine Dokumentation der API. In diesem Kapitel wird ein Überblick über die Funktionalität und einige Entwurfsentscheidungen gegeben.

5.1. Kommandozeileninterface

Die folgenden Programme sind enthalten:

`produce-features`
: Liest Signal- oder Positionsdaten und produziert eine Merkmalsdatei. Die Datei enthält für jede Sekunde und für jedes Gerätepaar einen Merkmalsvektor. Unterstützte Algorithmen sind `dtw`, `multi-dtw` und `euclid`. *time-lag* und *window-size* können konfiguriert werden.

`produce-ground-truth`
: Liest die *ground truth* einer Szene und erzeugt eine *ground-truth*-Datei, die für die Evaluation und das Training eines Klassifizierers benötigt wird. Eine solche Datei speichert für jeden Zeitpunkt eine Menge von Gruppen. Sofern vorhanden, wird auch die Reihenfolge von Personen in den Gruppen gespeichert.

`train-classifier`
: Trainiert einen Klassifizierer mit Merkmalsdaten und dazugehöriger *ground-truth*. Erzeugt eine Datei, die den gesamten Status des Klassifizierers speichert, sodass dieser wiederverwendet werden kann. Der Klassifizierer kann entscheiden, ob ein Merkmalsvektor zu einem Gerätepaar gehört, welches *co-moving* ist.

detect-followers

Benötigt als Input einen Klassifizierer und eine beliebige Merkmalsdatei. Findet sich gemeinsam bewegende Gerätepaare mithilfe des Klassifizierers und entscheidet dann die Anführer-/Nachfolgerbeziehung dieser Paare. Erzeugt eine Datei, die diese Information für jeden Zeitpunkt speichert.

detect-leaders

Liest eine Datei, die von `detect-followers` produziert wurde und erzeugt eine Datei, welche für jeden Zeitpunkt die Menge der Anführer speichert.

Zusätzlich gibt es mehrere Programme, die nur der Evaluation dienen. Jedes dieser Programme verfügt außerdem über die Optionen `-h` bzw. `--help`.

Beispiel

```
# Erzeugt Merkmalsdatei 'scene.feature'. 'manifest.json' speichert
# Metadaten über die Szene (e.g. Pfad zu Messungen). Ausgabetyp ist 'json'.
# Ausgabetyp 'binary' ist effizienter, aber unleserlich.
$ produce-features --input manifest.json \
    --output scene.feature --output-type json \
    --algorithm dtw --window-size 15 --time-lag 7 --threads 4
$ produce-ground-truth --input manifest.json --output scene.ground-truth
# In diesem vereinfachten Beispiel wird der Klassifizierer mit den
# Daten trainiert, die er auch entscheiden soll.
# In der Praxis können die Trainingsdaten aus anderer Quelle stammen.
$ train-classifier --input scene.feature --input-type json \
    --ground-truth scene.ground-truth --output scene.classifier
$ detect-followers --classifier scene.classifier \
    --input scene.feature --input-type json --output scene.followers
$ detect-leaders --input scene.followers --output scene.leaders
```

5.2. Bibliothek

Die Bibliothek `libmp` („mp” für *movement patterns*) implementiert den Algorithmus Schritt für Schritt als eine Serie von Transformationen, die auf Datensätze angewandt werden. Für jeden dieser Schritte folgt eine Übersicht. Der Quellcode der Bibliothek befindet sich im `namespace mp`.

Rohdaten

Daten betreten das System entweder als Instanzen von `signal_data` (für Signalstärkenmessungen) oder `location_data` (für räumliche Koordinaten). Die Struktur `signal_data` enthält für jedes Gerät eine Liste von Messungen. Jede dieser Messungen gehört zu einem Zeitpunkt und einem Access Point und gibt dessen Signalstärke an. Die Anzahl von Messungen pro Zeitpunkt ist variabel, da die Anzahl der von einem Gerät „sichtbaren" Access Points in der Regel schwankt. `location_data` enthält ebenfalls eine Liste von Messungen pro Gerät; hier speichert jede Messung direkt den gesamten Koordiantensatz zu einem bestimmten Zeitpunkt. Die von der Implementierung benutzte Zeiteinheit ist 1 Sekunde.

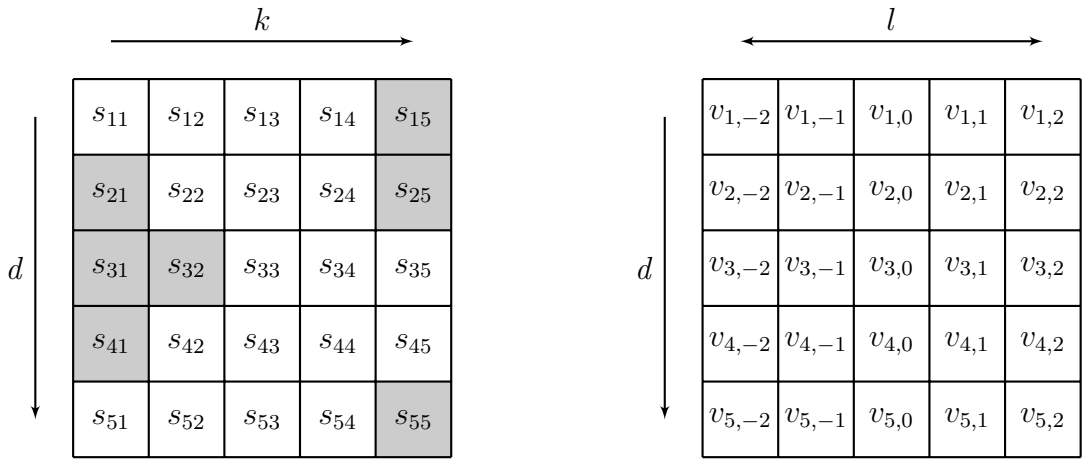
Die Bibliothek verfügt über die Funktionalität, „schlechte" Access Points aus den Signaldaten zu entfernen. Als „schlecht" gilt ein Access Point dann, wenn dessen durchschnittliche Signalstärke unter einem benutzerdefinierten Wert liegt. In der Praxis könnte ein solcher Access Point beispielsweise zu einem benachbarten Gebäude gehören oder defekt sein, sodass dessen Daten nicht zuverlässig sind. Um Spitzen in den Messungen zu kompensieren, werden Signaldaten außerdem durch ein *moving average* mit konfigurierbarer Länge geglättet. [1]

Es werden mehrere `parse`-Funktionen zur Verfügung gestellt, die Signaldaten oder räumliche Koordinaten aus CSV-Dateien einlesen und `signal_data` oder `location_data` Instanzen zurückgeben. Diese Strukturen können außerdem programmatisch erstellt werden, sofern deren Invarianten eingehalten werden.

Die Struktur `tracing_data` implementiert die Verallgemeinerung, die in Kapitel 4 durch die Menge M und die Messdimension k stattfindet. Für jedes Gerät wird eine Matrix von Messdaten angelegt. Diese Matrix hat die Größe $d \times k$ und speichert in jeder Zeile den gesamten Messvektor (d ist die Dauer). Parallel dazu wird vermerkt, welche Daten bei der Erstellung der Matrix vorhanden waren und welche durch einen Standardwert zustandekamen (dies entspricht der Indexmenge I und dem Wert ε aus Kapitel 4). Für Signaldaten ist k die Anzahl aller bekannten Access Points; zu einem Zeitpunkt fehlende Werte werden durch den konfigurierbaren Standardwert[2] ersetzt und entsprechend vermerkt. Für Positionsdaten ist $k = 3$. Die Organisation der Daten auf diese Weise hat den

[1] `produce-features` benutzt -90 dbm als untere Grenze für die durchschnittliche Signalstärke. In der Evaluation werden Signaldaten über einen Zeitraum von 5 s geglättet.

[2] In `produce-features` ist dies -100 dbm.



(a) Messwerte in `tracing_data` zu einem einzelnen Gerät.

(b) Merkmalsvektoren in `similarity_data` zu einem Gerätepaar.

Abbildung 5.1.: Schematische Darstellung der Datenstrukturen.

Vorteil, dass im Folgenden nur ein gemeinsamer Datentyp betrachtet werden muss. Da die Daten als Matrix im benachbarten Speicher angeordnet sind, ist die räumliche Lokalität garantiert. Dies spielt bei der späteren Berechnung der Merkmalsdaten eine Rolle, da dort zeilenweise über diese Daten iteriert wird. [3]

Abbildung 5.1a zeigt die resultierende Datenstruktur für ein einzelnes Gerät. Fehlende Messungen, die durch den Standardwert ersetzt wurden, sind grau gekennzeichnet. Element s_{ij} ist die Signalstärke zum Access Point j und zum Zeitpunkt i.

Merkmalsdaten

Der nächste Schritt ist die Berechnung der Merkmalsdaten. Dies geschieht mit der Hilfe der Klasse `feature_computation`. Unterstützte Algorithmen sind `euclid`, `dtw` und `multi-dtw`. Die verschiedenen Ähnlichkeitsalgorithmen sind mithilfe von *compile-time*-Polymorphismus durch `template`-Klassen implementiert, damit die gemeinsamen Codeabschnitte ohne Leistungseinbußen geteilt werden können. Zusätzlich zu den aus Kapitel 4 bekannten Parametern *max-time-lag* und *window size* wird *Multithreading* unterstützt; die Anzahl der verwendeten Threads kann vom Benutzer festgelegt werden. [4] Die Parallelisierung ist möglich, weil die Berechnungen der Merkmalsdaten einzelner Paare vollkommen

[3] Eine Speicherung als Hashtable o.Ä. würde zwar den benötigten Speicher reduzieren (fehlende Werte), gleichzeitig aber auch den Zugriff um ein Vielfaches verlangsamen (wiederholtes Hashing).

[4] `produce-features` verfügt über eine Option, die als Standardwert die Anzahl der CPU-Kerne hat.

unabhängig voneinander sind. Die betrachteten N Paare werden in gleichmäßig großen Gruppen auf t Threads verteilt, sodass jeder Thread die Merkmalsdaten von ungefähr $\frac{N}{t}$ Paaren berechnet. Für sehr große Datenmengen wird eine weitere Parellelisierung bezüglich der Zeit sinnvoll sein, da auch die einzelnen Zeitpunkte unabhängig voneinander berechnet werden können. Bei den in dieser Arbeit vorkommenden Problemgrößen ist eine weitere Optimierung allerdings unnötig (siehe Kapitel 6).

`libmp` verfügt über eine eigene Implementation von Dynamic Time Warping und der euklidischen Distanz. DTW ist vollkommen generisch implementiert, sodass der Algorithmus sowohl auf Folgen von eindimensionalen Werten (`dtw`) als auch auf mehrdimensionale Folgen (`multi-dtw`) anwendbar ist. An der Aufrufsstelle muss daher auch sichergestellt werden, dass eine passende *lokale Kostenfunktion* zur Verfügung gestellt wird.

Bei der Berechnung der Ähnlichkeit von zwei Folgen durch DTW wird der Warp-Pfad nicht benötigt. Lediglich dessen Kosten gehen in die Merkmalsvektoren ein. Die Kostenmatrix C, die von DTW intern genutzt wird, kann also wiederverwendet werden. Um die Evaluation zu unterstützen, kann der *warping path* trotzdem berechnet werden, indem der aktuelle Inhalt der Kostenmatrix betrachtet wird. Eine erneute Ausführung von DTW überschreibt den Inhalt. Durch diese Optimierung findet die Berechnung einzelner Ähnlichkeitswerte ohne jede Speicherallokation statt (diese erfolgt t mal zu Beginn, da jeder Thread eine private Matrix benötigt).

Die resultierenden Merkmalsdaten werden in der Struktur `similarity_data` zurückgegeben. Hier wird für jedes betrachtete Gerätepaar eine Matrix gespeichert, die für jeden Zeitpunkt (Zeilen) einen *feature vector* enthält. Der *feature vector* wiederum enthält für jeden *time-lag* (Spalten) den Ähnlichkeitswert. Abbildung 5.1b zeigt eine solche Matrix: v_{ij} ist der Ähnlichkeitswert an Zeitpunkt i für *time-lag* j.

In Sektion 3.1 wurden mehrere Strategien vorgestellt, um die Laufzeit des DTW-Algorithmus zu verbessern. Dynamic Time Warping benötigt für Folgen gleicher Länge quadratische Zeit; in diesem Kontext ist die Laufzeit daher quadratisch zur *window size*. Kjærgaard und Blunck haben in ihrer Arbeit für die Fenstergröße Werte zwischen 5 und 15 Sekunden gewählt [KB14]. Da als Zeiteinheit 1 Sekunde benutzt wird, ist die Größe einer jeden berechneten Kostenmatrix für DTW maximal 15×15. Bei einer so kleinen Problemgröße ist zu erwarten, dass fortgeschrittene Verfahren wie *FastDTW* oder *SparseDTW* aufgrund ihres Overheads zu langsamerer Ausführung führen.

```
[{
    left: 3, right: 5,
    lag: -1.5, type: following
},
{
    left: 5, right: 2,
    lag: 0.7, type: leading,
},
{
    left: 4, right: 5,
    lag: -0.3, type: following
}, ... ]
```

(a) Beziehungen in `following_data` zu einem bestimmten Zeitpunkt.

```
[{
    timestamp: 0,
    leaders: [1, 3]
},
{
    timestamp: 1,
    leaders: [1, 3, 4]
},
{
    timestamp: 2,
    leaders: [3, 4]
}, ... ]
```

(b) Eine Liste von Gruppenanführern für jeden Zeitpunkt (`leader_data`).

Abbildung 5.2.: Schematische Darstellung der Datenstrukturen.

Abbildung 3.2 auf Seite 20 zeigte schon die Anhäufungen von Warp-Pfaden. Die dort visualisierten Daten wurden mithilfe der hier implementierten Software gewonnen (für drei verschiedene Eingaben). Dabei wurde jeder Pfad beachtet, ungeachtet der Beziehung, die zwischen den jeweiligen Geräten herrschte. Aus der Darstellung wird ersichtlich, dass ein Sakoe-Chiba Band für diesen Algorithmus zu guten Resultaten führen kann. Allerdings verringert ein solches Band in jedem Fall trotzdem die Präzision. Da die Implementierung für die betrachteten Problemgrößen sehr schnell ist, wurde die Präzision betreffend kein Kompromiss eingegangen.

Erkennung von Gruppenverhalten

Die Klasse `co_moving_classifier` ist für die Klassifizierung von Merkmalsvektoren als *co-moving* zuständig. Eine Instanz kann mit `similarity_data` und `ground_truth` trainiert werden und anschließend zur Klassifizierung weiterer Merkmalsvektoren genutzt werden. Die Implementierung des maschinellen Lernens fand mithilfe der Bibliothek *dlib-ml* statt [Kin09]. Diese bietet im Vergleich zu der häufig benutzten *LIBSVM* [CL11] ein modernes C++-Interface und insbesondere die Möglichkeit, eine *support vector machine* und beliebige zusätzliche Daten zu serialisieren. Eine so serialisierte Instanz kann zu einem späteren Zeitpunkt wiederhergestellt werden.

Die Funktion `classify` benötigt als Eingabe einen Klassifizierer und eine Instanz von `similarity_data`. Ausgabe ist die Struktur `following_data`, welche jedem Zeitpunkt in den Merkmalsdaten eine Liste von Beziehungen zuordnet. Die möglichen Beziehungen entsprechen denen in Sektion 4.3 und enthalten jeweils die Geräte, zu denen sie gehören,

den geschätzen *time-lag* und den daraus resultierenden Typ der Beziehung (*following*, *leading* oder *co-leading*). Ein Beispiel befindet sich in Abbildung 5.2a: *left* und *right* sind die referenzierten Geräte – eine Kante zeigt von „links" nach „rechts". Mithilfe der Funktion `following_graph_at` wird mit diesen Beziehungen ein Graph zu einem bestimmten Zeitpunkt erstellt. Dies geschieht unter Zuhilfenahme der *Boost Graph Library* (BGL) [SLL02], welche außerdem den für die Implementierung von Sektion 4.4 nötigen Algorithmus `connected_components` bereitstellt.

Die Funktion `detect_leaders` ruft schließlich für jeden Zeitpunkt die Algorithmen `connected_components` und `page_rank` auf und berechnet so zu jeder erkannten Gruppe einen Anführer. `libmp` enthält eine Implementierung von PageRank (nach Vorbild der BGL), die die Gewichtung von Kanten unterstützt. Als *damping factor* wurde $d = 0.85$ festgelegt. Der Algorithmus terminiert bei einem erreichten relativen Fehler von 10^{-6} oder nach höchstens 500 Iterationen. Das Ergebnis ist eine Liste von Anführen für den betrachteten Zeitpunkt. Eine Instanz der Struktur `leader_data`, die für jeden Zeitpunkt eine solche Liste enthält, wird zurückgegeben. Abbildung 5.2b zeigt eine Beispielinstanz von `leader_data`.

6. Experimente

6.1. Beschreibung der Daten

Im Folgenden wird zunächst die in Kapitel 4 vorgestellte Methode mithilfe der in Kapitel 5 beschriebenen Implementierung evaluiert. Dies geschieht in mehreren Stufen, um ein Maß für die Qualität der einzelnen Teilschritte zu erhalten. Die zur Evaluation genutzten Rohdaten, d.h. Signalstärkenmessungen, räumliche Koordinatenspuren und die dazugehörige *ground-truth*, wurden von Kjærgaard und Blunck zur Verfügung gestellt und sind identisch mit denen, die in ihrer Arbeit genutzt wurden [KB14]. Um möglichst unabhängige Ergebnisse auf Grundlage der veröffentlichten Beschreibung zu erhalten, wurden ebenfalls vorliegende Zwischenergebnisse nicht verwendet. Nach der Betrachtung der einzelnen Stufen folgt eine Untersuchung der Laufzeit der Implementierung.

Die vorliegenden Daten stammen aus zwei Experimenten, die von Kjærgaard und Blunck durchgeführt worden sind. Das erste Experiment ist eine Folge geskripteter Szenen, in denen sich jeweils bis zu 9 Testpersonen[1] in verschiedenen Gruppenkonstellationen innerhalb eines Gebäudes bewegen, wobei zwei Etagen genutzt werden. Die Testpersonen sind mit einem Smartphone ausgestattet, welches die Signalstärken nahe liegender Access Points über den Zeitraum der Szenen aufzeichnet. Zu jedem Zeitpunkt befinden sich für jedes Endgerät im Mittel 10 bis 14 Access Points in Reichweite. Die Anzahl der insgesamt vorkommenden Access Points liegt – pro Szene – zwischen 30 und 140. Für die Szenen wurde anhand von manueller Beobachtungen das Gruppenverhalten der Testpersonen vermerkt. Jede Person wurde für jeden Zeitpunkt einer Gruppe zugeordet. Die Reihenfolge der Personen in diesen Gruppen wurde festgehalten, sodass anhand dieser *ground-truth* auf die Kriterien *co-moving* und *following* geschlossen werden kann. Die Anführer einer

[1] Es liegen für mehr Personen Rohdaten vor. Sie können aber nicht zu Testzwecken genutzt werden, da die dazugehörige *ground-truth* diese nicht erwähnt und somit die Ergebnisse nicht überprüft werden können.

Gruppe sind die Personen, die in der Gruppenreihenfolge an erster Stelle stehen (es sind mehrere möglich, siehe *co-leading* in Sektion 4.3).

Das zweite Experiment wurde in der Form eines ungeskripteten Spiels durchgeführt und fand in einem anderen Gebäude statt. Die zehn Teilnehmer sind jeweils entweder ein *evader* oder ein *follower*. Ein *follower* erhält Punkte, indem er einem *evader* folgt. Ein *evader* hat die gegenteilige Aufgabenstellung: er erhält Punkte, wenn kein *follower* ihm folgt. Außerdem wurde ein gelegentliches Wechseln der Ziele angeregt, indem ein *follower*, der für mehr als 45 s demselben *evader* folgt, Minuspunkte erhält. Die mittlere Anzahl der Access Points pro Gerät und Zeitpunkt beläuft sich für dieses Experiment auf etwa 16. Insgesamt sind ca. 70 Access Points vorhanden. Aufgrund der Natur des Spiels kann die *ground-truth* nur für gewisse Personenpaare eine Aussage über deren Gruppenverhalten machen. Da jeweils nur festgehalten wird, welcher *follower* welchem *evader* folgt, können nur (*follower, evader*)-Paare analysiert werden. Es fehlen also Aussagen über (*follower, follower*)- und (*evader, evader*)-Paare. Eine detailliertere Beschreibung der beiden Experimente befindet sich in der Arbeit von Kjærgaard und Blunck [KB14].

6.2. Einstufung als *co-moving*

Die Klassifizierung von Merkmalsvektoren als *co-moving* geschieht, indem eine *support vector machine* mit Merkmalsvektoren und dazugehöriger *ground-truth* trainiert wird. Diese SVM wird auf weitere Merkmalsdaten angewandt. Man erhält ein binäres Klassifizierungsproblem, bei dem vier Fälle von Bedeutung sind:

- *true positive* und *true negative*. Ein zu einem Paar von Personen gehörender Merkmalsvektor wurde korrekt klassifiziert.
- *false positive* (auch Fehler vom Typ I). Ein Merkmalsvektor wurde fälschlicherweise als *co-moving* eingestuft.
- *false negative* (auch Fehler vom Typ II). Ein Merkmalsvektor wurde fehlerhaft als nicht-*co-moving* erkannt.

Es wurden für mehrere Parameterkombinationen Merkmalsdaten aus den vorliegenden Rohdaten gewonnen. Die variablen Parameter sind die Ähnlichkeitsfunktion und die *window size*. Der maximale *time-lag* z ist hier und im Folgenden immer auf 7 s festlegt. Dies

entspricht dem Wert, der von Kjærgaard und Blunck bei der Auswertung aber insbesondere auch bei der Erstellung der *ground-truth* verwendet wurde: 7 s entsprechen einem maximalen Abstand von 7 m zwischen zwei sich gemeinsam bewegender Personen, setzt man eine Geschwindigkeit von 1 m/s voraus. Da die *ground-truth* auf Grundlage dieser Überlegung angefertigt wurde, kann kein anderer *time-lag* sinnvoll bewertet werden [KB14, Sektion 5].

Ein Klassifizierer wurde mit Merkmalsdaten und *ground-truth* einer der geskripteten Szenen trainiert und benutzt, um die Merkmalsdaten aller dieser Szenen zu klassifizieren. Analog lernte eine SVM aus einem Teil der Spieldaten und wurde auf alle Spieldaten angewandt. Die verwendete Kernelfunktion war in beiden Fällen linear. Tabelle 6.1 zeigt die Resultate für verschiedene Algorithmus- und *window size*-Kombinationen. Abgebildet sind sowohl die Korrektklassifikationsrate (Korrektheit) wie auch das F-Maß (*F score*). Die Korrektheit ist die relative Häufigkeit insgesamt korrekt zugeordneter Merkmalsvektoren, d.h. die relative Häufigkeit von *true positive* und *true negative* in Bezug zu allen Merkmalsvektoren. Bei den hier verwendeten Klassifizierern kam es häufig zu Fehlern vom Typ I, während die Sensitivität (rel. Häufigkeit von *als co-moving erkannt* in Bezug zu *tatsächlich co-moving*) sehr hoch war. Fehler vom Typ II traten nur in sehr geringer Anzahl auf. Die Menge der als *co-moving* erkannten Vektoren war also zu „grob". Diese Fälle schlagen sich im F-Maß nieder, welches als (gewichtetes) harmonisches Mittel von Sensitivität und Genauigkeit definiert ist. Die Genauigkeit (auch positiver Vorhersagewert) wiederum ist die rel. Häufigkeit der *true-positive*-Fälle unter allen als positiv eingestuften Vektoren. Ein niedriges F-Maß deutet in diesem Kontext also auf eine hohe Häufigkeit von *false-positives* hin.

Aus Tabelle 6.1 ist abzulesen, dass der DTW-Algorithmus in allen Szenarien die besseren Ergebnisse liefert. Dynamic Time Warping ist im Vergleich zur euklidischen Methode resistenter gegen Verzerrungen. Diese können z.B. durch Fluktuationen in den Messungen oder durch zeitliche Verzerrung der Muster (etwa durch die Geschwindigkeit eines Fußgängers) auftreten. Multi-DTW benutzt intern die euklidische Metrik als lokale Kostenfunktion, sodass die Nachteile dieses Ansatzes auch hier in Erscheinung treten. Dies wird dadurch bestätigt, dass die Korrektheit von Multi-DTW stets zwischen der von Euklid und DTW liegt. Das beste Ergebnis liegt für DTW und eine *window size* von 15 Sekunden vor.

Die Auswirkung der Fenstergröße w ist für für beide Experimente gut zu erkennen. Sie entspricht der Länge des für die Erzeugung der Ähnlichkeitswerte betrachteten Zeitraums.

Tabelle 6.1.: Evaluation eines *co-moving classifiers*.
window size w in Sekunden. Korrektheit und F-Maß in Prozent.

(a) Ergebnisse für geskriptete Szenen.

		Signalstärkedaten		Positionsdaten	
Algorithmus	w	korrekt	F-Maß	korrekt	F-Maß
Euklid	5	91,59	88,51	84,61	80,15
	10	91,95	88,93	85,05	80,61
	15	92,56	89,64	85,71	81,25
DTW	5	90,84	87,74	85,85	81,58
	10	91,82	88,87	86,86	82,59
	15	93,04	90,33	88,24	84,09
Multi-DTW	5	91,59	88,51	84,74	80,28
	10	91,96	88,95	85,71	81,27
	15	92,69	89,80	86,88	82,45

(b) Auswertung mit ungeskripteten Spieldaten.

		Signalstärkedaten		Positionsdaten	
Algorithmus	w	korrekt	F-Maß	korrekt	F-Maß
Euklid	5	76,55	59,83	76,12	57,85
	10	76,45	59,81	76,24	58,56
	15	76,74	60,29	76,92	59,99
DTW	5	76,60	59,98	78,92	60,10
	10	76,64	60,17	78,02	60,68
	15	76,98	60,68	77,56	61,03
Multi-DTW	5	76,58	59,86	76,23	58,30
	10	76,47	59,83	75,87	58,90
	15	76,80	60,36	75,97	59,58

Ein größerer Zeitraum kann zu besseren Ergebnissen führen, sofern die Bewegungsmuster eine dafür angemessene Länge aufweisen. Bei einer zu großen *window size* kann es vorkommen, dass ähnliche Abschnitte der Muster zu kurz sind, um das Ergebnis zu beeinflussen.

Ein Probelauf mit dem ungeskripteten Datensatz und einer *window size* von 45 s lieferte deutlich schlechtere Resultate. Dies entspricht dem Zeitraum, ab dem Minuspunkte vergeben wurden, wenn ein *follower* immer dem gleichen *evader* folgte.

Insgesamt werden sich gemeinsam bewegende Paare auf der Grundlage von Signalstärkedaten besser identifiziert. Dies kann mit der geringen Dimension der räumlichen Koordinaten erkärt werden (im Vergleich zur Anzahl der nahen Access Points). Bei Positionsdaten mussten die Bewegungsmuster anhand von nur 3 Werten pro Zeitpunkt erkannt werden, während für Signalstärken deutlich mehr Daten zur Verfügung standen.

Sowohl bei der Auswertung der geskripteten wie auch der ungeskripteten Szenen finden sich bezüglich Algorithmus, *window size* und Art der Daten die gleichen Trends. Trotzdem sind die Resultate für die Spielszenarien deutlich schlechter. Insbesondere die Anzahl der Fehler vom Typ I ist besonders hoch, die Ursache dafür ist allerdings unklar.

Zusätzlich zu den in Tabelle 6.1 aufgeführten Szenarien wurde eine Kreuzvalidierung durchgeführt. Ein mit den Merkmalsdaten des Spiels trainierter Klassifizierer wurde für die geskripteten Szenen eingesetzt und umgekehrt. Die erhaltenen Daten unterscheiden sich von den hier abgebildeten im Wesentlichen nicht. Die Ähnlichkeit der Resultate ist ein Indiz dafür, dass die hier untersuchten Bewegungsmuster unabhängig sind von der Umgebung, in der sie gesammelt wurden (z.B. Anordnung der Access Points, Gebäudestruktur). Außerdem scheinen sie nicht von den konkreten Gruppenbeziehungen abhängig zu sein.

6.3. Beziehungen zwischen Paaren

Um die Wirksamkeit der Methode aus Sektion 4.3 zu überprüfen, wurden die *co-moving* eingestufen Paare mithilfe der Implementierung der Nachfolgererkennung untersucht. Aus dem geschätzten *time-lag* ergibt sich für die Personen jedes Paares entweder die Einstufung als Anführer oder als Nachfolger. Wieder wurden die besten Ergebnisse für eine *window size* von 15 s erhalten. Die Erfolgsraten dieser Klassifizierung sind in Abbildung 6.1 dargestellt.

Da es bei der Einstufung als *co-moving* insbesondere für die ungeskripteten Szenen eine hohe Anzahl von Fehlern vom Typ I (*false positive*) gab, wurden die Daten auf zwei verschiedene Arten untersucht. Zum einen wurden alle als *co-moving* klassifizierten Paare

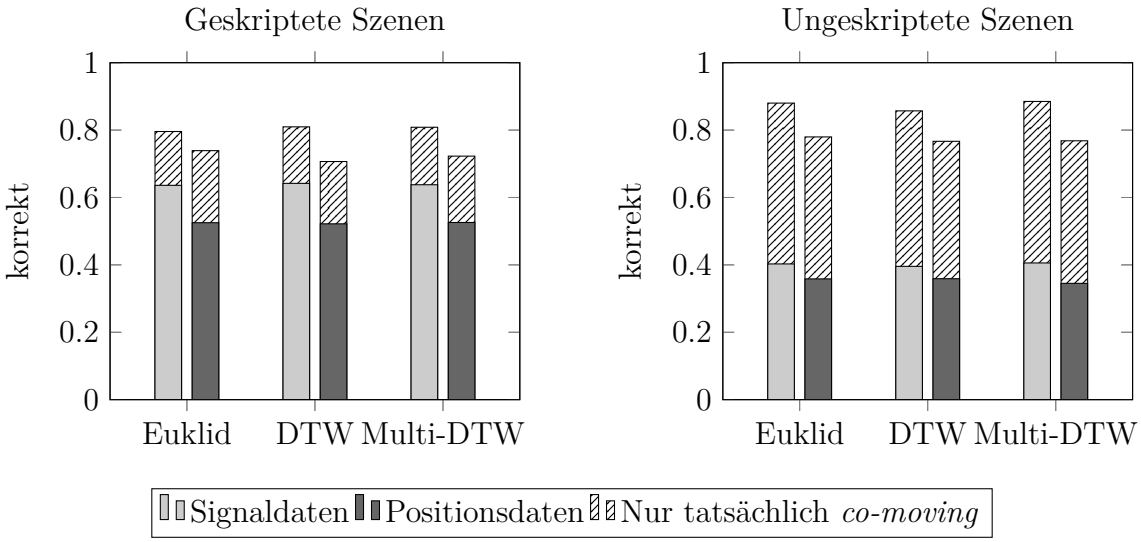


Abbildung 6.1.: Ergebnisse für Beziehungen zwischen sich gemeinsam bewegenden Paaren für eine *window size* von 15 s auf Grundlage von Signal- und Positionsdaten.

betrachtet (Ergebnisse als graue Balken in Abbildung 6.1). Um eine gute Aussage über die Schätzmethode aus Sektion 4.3 zu erhalten, wurden zum anderen die Paare mithilfe der *ground truth* reduziert, sodass nur solche betrachtet wurden, die tatsächlich *co-moving* waren. Die Erfolgsraten für diese Paare sind als gestrichelte Bereiche dargstellt. Man kann ablesen, dass die Schätzmethode mit einer mittleren Erfolgsrate von etwa 80% gute Ergebnisse erziehlt. Die insgesamt geringere Qualität der Resultate lässt sich also auf fehlerhafte Klassifizierungen als *co-moving* zurückführen.

6.4. Gruppenanführer

Die Erkennung von Gruppenanführern (siehe Sektion 4.4) ist wieder ein binäres Klassifizierungsproblem. Der PageRank-Algorithmus wurde auf die ungefilterten Beziehungen aus der vorherigen Sektion angewandt. Für Merkmalsdaten, die bei einer *window size* von 15 s unter Benutzung des DTW-Algorithmus gewonnen wurden, sind die Ergebnisse in Tabelle 6.3 aufgeführt. Da für jede Gruppe nur ein Anführer erkannt wird, ist die relative Anzahl der Anführer im Vergleich zu allen Personen offensichtlich klein. Deshalb wird in der Tabelle neben Korrektklassifikationsrate und F-Maß auch die Sensitivität genannt (*true positive rate*), welche Aufschluss darüber gibt, welcher Anteil der Anführer tatsächlich erkannt wurde.

Tabelle 6.3.: Auswertung der Anführererkennung (Algorithmus: DTW, *window size*: 15 s). Sensitivität, Korrektheit und F-Maß in Prozent.

		Signalstärkedaten			Positionsdaten		
Datentyp	Variante	Sens.	korrekt	F-Maß	Sens.	korrekt	F-Maß
geskriptet	ungew.	74,93	84,98	77,70	62,64	72,54	65,32
	gew.	79,29	87,95	82,23	63,50	73,28	66,20
ungeskriptet	ungew.	40,76	61,43	52,24	45,90	65,31	57,84
	gew.	41,07	61,95	52,87	43,72	63,02	55,18

Es wurden für Positions- und Signaldaten jeweils die ungewichtete und die gewichtete Variante von PageRank überprüft. Die gewichtete Variante erzielte für das geskriptete Experiment bessere Resultate, insbesondere für Signalstärkedaten. Anführer wurden mit einer Sensitivität von 79% identifiziert. Die Ergebnisse des ungeskripteten Spiels sind hier, wie auch in der letzten Sektion, deutlich schlechter. Dies hängt mit der weniger guten Einstufung als *co-moving* zusammen, wodurch auch die Korrektheit der erkannten Beziehungen verringert wird.

Eine weiteres Problem bei der Erkennung von Anführern ist die Verwendung des *connected-components*-Algorithmus. Dieser liefert die Menge der untereinander verbundenen Teilgraphen; jeder dieser Teilgraphen wird als Gruppe angesehen. Hier benötigt es nur eine fehlerhafte Kante, die ansonsten nicht verbundene Teilgraphen zusammenschließt, sodass diese als nur eine Gruppe erkannt werden. In den Experimenten trat dieses Szenario häufig auf: schon die Anzahl der erkannten Anführer war falsch, nicht nur deren Identität.

Abbildung 6.2 zeigt den Gruppengraphen einer geskripteten Szene zu einem festen Zeitpunkt. Die Daten wurden mit dem Programm `following-graph` ausgelesen und manuell visualisiert. Zu dem abgebildeten Zeitpunkt war die Erkennung von Beziehungen und Anführern fehlerlos.

In Abbildung 6.3 sind erkannte Gruppen und ihre Anführer zu einem festen Zeitpunkt dargestellt. Die Kreise repräsentieren die einzelnen Gruppenmitglieder; ein schwarzer Kreis ist der Anführer der Gruppe. Eine Linie zwischen den Kreisen stellt ein erkanntes *following pattern* dar. Gruppen sind mit gestrichelten Linen voneinander abgegrenzt. Die Positionen der Kreise sind die tatsächlichen x- und y-Koordinaten der Personen (sie bewegen sich in der gleichen Etage). Die Daten stammen aus einer der ungeskripteten Szenen. Es ist sehr

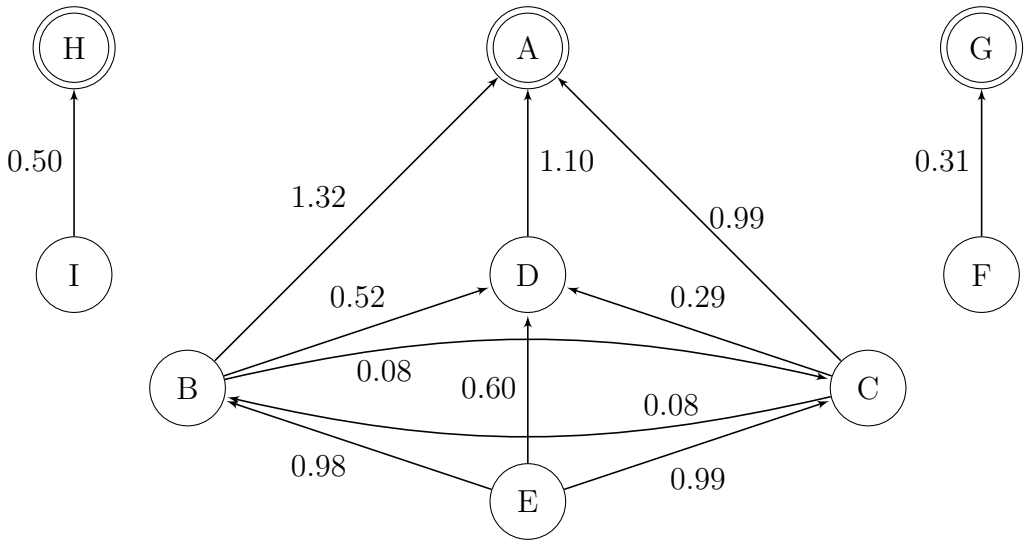


Abbildung 6.2.: Ein Gruppengraph (s.a. Sektion 4.4). Eine gerichtete Kante bedeutet „folgt". Die Kantengewichte entsprechen dem Betrag des geschätzten *time-lags*. Mit PageRank gefundene Anführer sind durch einen zweifachen Kreis gekennzeichnet.

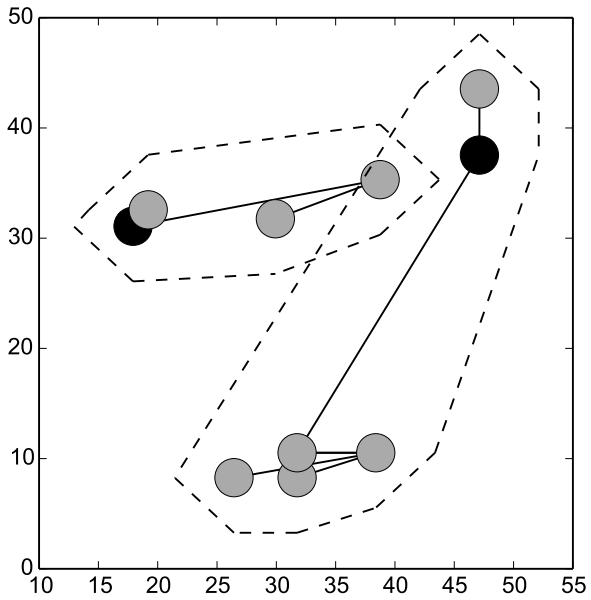


Abbildung 6.3.: Gruppenmitglieder und ihre tatsächlichen räumlichen Positionen.

gut zu erkennen, dass aus räumlicher Nähe nicht unbedingt ein *following pattern* folgen muss.

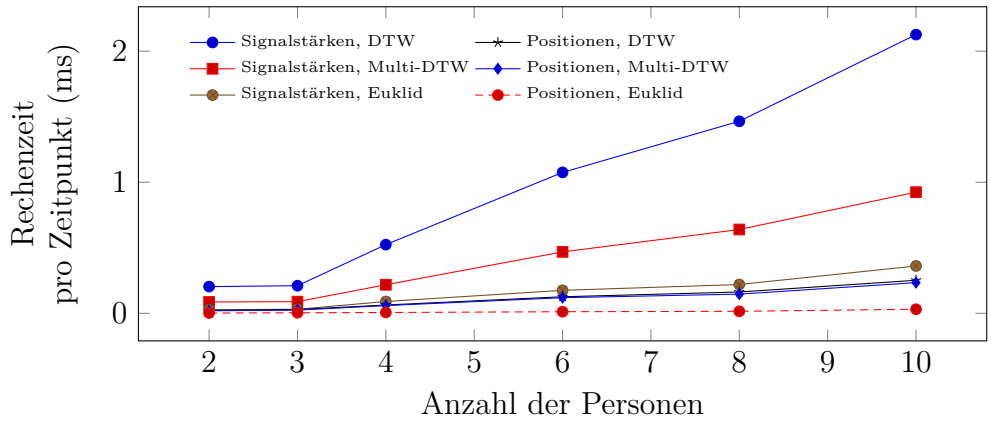


Abbildung 6.4.: Für die Berechnung der Merkmalsvektoren benötigte Zeit in Abhängigkeit von der Anzahl der Personen, pro betrachtetem Zeitpunkt.

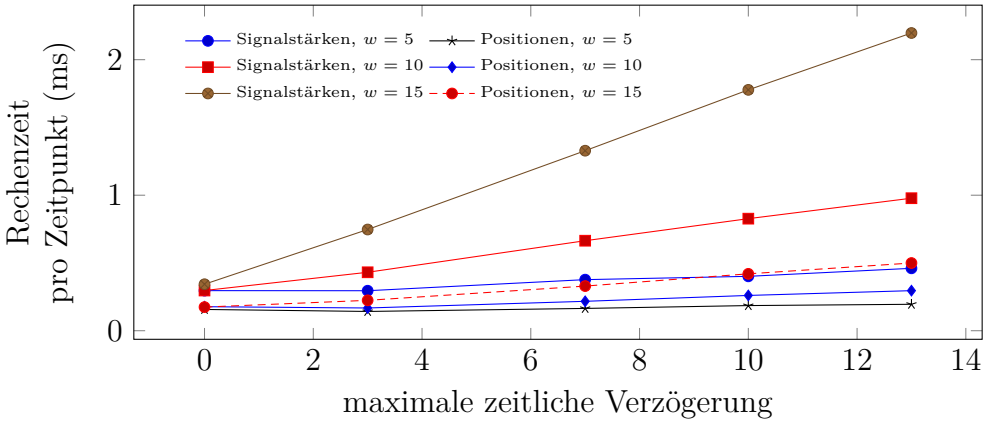


Abbildung 6.5.: Benötigte Zeit für die Durchführung aller Schritte in Abhängigkeit vom maximalen *time-lag*, pro betrachtetem Zeitpunkt. Es wurden 9 Geräte betrachtet.

6.5. Leistung

Abbildungen 6.4 und 6.5 visualisieren die benötigte Laufzeit für verschiedene Parameter. Die Daten wurden auf einem Desktop-Computer mit einem 3.4 GHz Intel Core i5 Prozessor und 16 GB RAM gesammelt. Abbildung 6.4 zeigt die Zeit, die für die Erstellung der Merkmalsdaten bei einem maximalen *time-lag* z von 7 s und einer *window size* w von 15 s benötigt wurde. Es ist deutlich zu sehen, dass der DTW-Algorithmus sowohl für Signalstärken wie auch für räumliche Daten am längsten arbeitet. Die euklidische Variante terminiert am schnellsten - dies lässt sich mit der asymptotischen Laufzeit erklären, die für die euklidische Ähnlichkeit im Unterschied zu den anderen beiden Verfahren nicht quadratisch in w ist. Die Laufzeit des Multi-DTW-Algorithmus liegt in beiden Fällen

zwischen DTW und Euklid, er stellt also sowohl für die Leistung wie auch die Präzision einen Mittelweg dar.

In Abbildung 6.5 können die Auswirkungen der maximalen Verzögerung und der Fenstergröße abgelesen werden. Es wurde die Laufzeit für alle Schritte (d.h. Merkmalsdaten, *co-moving*- und Nachfolgererkennung, Gruppenanführer) unter Verwendung des DTW-Algorithmus gemessen. Laut Analyse in Sektion 4.1 ist die asymptotische Laufzeit – bei ansonsten festen Parametern – linear im maximalen *time-lag* z; dies geht auch aus der Abbildung hervor. Auch hier tritt wieder die große Diskrepanz zwischen Signalstärkedaten und Positionsdaten auf. Diese folgt unmittelbar aus der deutlich geringeren Dimension der Eingabedaten. Kjærgaard und Blunck erhielten das Laufzeitverhalten betreffend für ihre Implementierung sehr ähnliche Ergebnisse [KB14, Fig. 15].

7. Zusammenfassung

Die Evaluation des Verfahrens hat gezeigt, dass die Methode die Bewegungsmuster von Personen mit zufriedenstellender Präzision erkennen kann. Mit einer Korrektklassifikationsrate von bis zu 93% bei der Einstufung als *co-moving* sind die Ergebnisse für Berechnungen auf der Grundlage von Signalstärken besonders gut. Gute Ergebnisse wurden auch bei der Erkennung von Anführer-/Nachfolgerbeziehungen und der Erkennung von Gruppenanführern erhalten: Mit einer Korrektklassifikationsrate von 88% wurde der richtige Anführer einer Gruppe gefunden. Diese Zahlen für das geskriptete Experiment sind vergleichbar mit denen, die von Kjærgaard und Blunck publiziert wurden [KB14].

Die Ursache für die deutlich schlechteren Ergebnisse im Falle des ungeskripteten Experiments ist unklar. Besonders problematisch war die Klassifizierung als *co-moving* mit einer sehr hohen *false-positive*-Rate. Aufgrund dieser Fehler vom Typ I wurden Anführer-/Nachfolgerbeziehungen falsch erkannt, sodass schließlich auch die Detektion von Gruppenanführern in vielen Fällen versagte.

Trotz dieser Fehlschläge ergeben sich zu beiden Datensätzen dieselben Trends: Die besten Ergebnisse werden für die DTW-Methode bei einer *window-size* von 15 s erhalten. Die in dieser Arbeit neu hinzugefügte Multi-DTW-Methode erzielte in allen Szenarien schlechtere Resultate, sodass die Benutzung der DTW-Methode empfehlenswert ist.

Bei der Implementierung der Software wurde auf hohe Leistungsfähigkeit geachtet. Mit einer Rechenzeit von etwa 1.5 Millisekunden pro analysiertem Zeitpunkt (bei 9 Geräten) ist die Software dazu geeignet, auf deutlich größere Datenmengen angewendet zu werden. Eine Steigerung der zeitlichen Auflösung könnte die Genauigkeit verbessern. Denkbar sind auch Echtzeitanwendungen, für die schnelle Reaktionsfähigkeit von großer Wichtigkeit ist.

Es gibt mehrere Ansätze, mit denen das Verfahren verfeinert werden kann. Bei der Implementierung der *co-moving*-Klassifizierung erwies sich die Benutzung einer *support vector*

machine zumindest für einen der Datensätze als sehr problematisch. Eine Verbessung der Korrektklassifikationsrate würde in allen anderen Schritten zu deutlich gesteigerter Präzision führen. Außerdem zeigte sich, dass die Verwendung des *connected-components*-Algorithmus für die Erkennung von Gruppen im Angesicht fehlerhaft erkannter Beziehungen nicht robust genug ist.

Als weiterer Ansatz können Daten aus anderen Quellen in Betracht gezogen werden. Beispielsweise ließen sich die elektronischen Pläne eines Gebäudes dazu nutzen, gemeinsame Bewegungsmuster von Paaren auszuschließen; etwa wegen trennender Wände oder anderer Hindernisse. Außerdem kann in einigen Anwendungsbereichen *a priori* Wissen genutzt werden, um als Heuristik für Gruppenverhalten zu dienen: In einem Computerspiel mit sozialen Elementen würde z.B. erwartet werden, dass ein Spieler häufig mit den gleichen Personen – etwa seinem Freundeskreis – interagiert.

Abbildungsverzeichnis

Alle Abbildungen wurden selbstständig erstellt.

Literaturverzeichnis

[ACT12] Al-Naymat, Ghazi, Sanjay Chawla und Javid Taheri: *SparseDTW: A Novel Approach to Speed up Dynamic Time Warping.* CoRR, abs/1201.2969, 2012.

[AVM+12] Albanese, Davide, Roberto Visintainer, Stefano Merler, Samantha Riccadonna, Giuseppe Jurman und Cesare Furlanello: *mlpy: Machine Learning Python.* CoRR, abs/1202.6548, 2012. `http://mlpy.sourceforge.net`.

[BP98] Brin, Sergey und Lawrence Page: *The anatomy of a large-scale hypertextual Web search engine.* Computer Networks and ISDN Systems, 30(1–7):107 – 117, 1998. Proceedings of the Seventh International World Wide Web Conference.

[BZ04] Bhanu, B. und Xiaoli Zhou: *Face recognition from face profile using dynamic time warping.* In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, Band 4, Seiten 499–502 Vol.4, Aug 2004.

[CL11] Chang, Chih-Chung und Chih-Jen Lin: *LIBSVM: A library for support vector machines.* ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[DWL08] Dodge, Somayeh, Robert Weibel und Anna-Katharina Lautenschütz: *Towards a Taxonomy of Movement Patterns.* Information Visualization, 7(3):240–252, Juni 2008.

[KB14] Kjærgaard, Mikkel Baun und Henrik Blunck: *Tool support for detection and analysis of following and leadership behavior of pedestrians from mobile sensing data.* Pervasive and Mobile Computing, 10, Part A(0):104 –

117, 2014. Selected Papers from the Eleventh Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2013).

[Kin09] King, Davis E.: *Dlib-ml: A Machine Learning Toolkit*. Journal of Machine Learning Research, 10:1755–1758, 2009. http://dlib.net/ml.html.

[Kjæ10] Kjærgaard, Mikkel Baun: *Indoor Positioning with Radio Location Fingerprinting*. CoRR, abs/1004.4759, 2010.

[KP00] Keogh, Eamonn J. und Michael J. Pazzani: *Scaling Up Dynamic Time Warping for Datamining Applications*. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, Seiten 285–289, New York, NY, USA, 2000. ACM.

[Mü07] Müller, Meinard: *Information Retrieval for Music and Motion*. Springer-Verlag, Berlin Heidelberg, 2007.

[PBMW99] Page, Lawrence, Sergey Brin, Rajeev Motwani und Terry Winograd: *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66, Stanford InfoLab, November 1999.

[RCM+12] Rakthanmanon, Thanawin, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria und Eamonn Keogh: *Searching and Mining Trillions of Time Series Subsequences Under Dynamic Time Warping*. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, Seiten 262–270, New York, NY, USA, 2012. ACM.

[SC78] Sakoe, H. und S. Chiba: *Dynamic programming algorithm optimization for spoken word recognition*. Acoustics, Speech and Signal Processing, IEEE Transactions on, 26(1):43–49, Feb 1978.

[SC07] Salvador, Stan und Philip Chan: *Toward Accurate Dynamic Time Warping in Linear Time and Space*. Intelligent Data Analysis, 11(5):561–580, Oktober 2007.

[SLL02] Siek, Jeremy, Lie-Quan Lee und Andrew Lumsdaine: *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. http://www.boost.org/doc/libs/1_57_0/libs/graph/doc/.

A. Inhaltsverzeichnis der DVD

```
DVD
|-- build/                      -- Enthält kompilierte ausführbare Dateien (1)
|-- data/                       -- Rohdaten (Signalstärken, ground-truth, ...)
|-- doc/                        -- API-Dokumentation des C++ Projekts (2)
|-- mp/                         -- C++ Projekt (Bibliothek und Kommandozeile)
|   |-- deps/                   -- Mitgelieferte benutzte Bibliotheken
|   |-- include/                -- C++-Headerdateien von libmp
|   |-- src/                    -- Quellcode von libmp und Kommandozeilentools
|   |-- test/                   -- Quellcode der Unit-Tests
|   `-- CMakeLists.txt          -- CMake Projektdatei
|-- output/                     -- Enthalt rohe Ausgabe der Programme (3)
|-- results/                    -- Aufbearbeitete Resultate der Skripte (4)
|-- scripts/                    -- Quellcode der Skripte
|-- build.sh                    -- Kompiliert den C++-Code
|-- produce-results.sh          -- Erzeugt Inhalte von results/ und output/ (5)
|-- Bachelorarbeit.pdf          -- Elektronische Version dieses Dokumentes
`-- README                      -- Technische Informationen
```

(1) Die Programme in `build/` sind in Kapitel 5 kurz beschrieben. Sie werden von dem Skript `build.sh` erzeugt.

(2) Mit dem Browser `doc/html/index.html` öffnen.

(3) Enthält die rohe Ausgabe der Kommandozeilentools. Für jede Kombination der Eingabeparameter (window size, time lag, Algorithmus) wird für die jeweile Kategorie (Merkmalsdaten etc.) ein Ordner erstellt. Diese Ordner enthalten für jede betrachtete Szene verschiedene Dateien, im Fall der Merkmalsdaten sind dies z.B. die Merkmalsdaten selbst, damit trainierte Klassifizierer und deren Evaluationsergebnisse.

(4) In `results/` sind die aufbereiteten Ergebnisse gespeichert (als Texte oder Grafiken).

(5) Dieses Skript erzeugt mit der Hilfe der Python-Skripte in `scripts/` den oben genannten Output und die Resultate. Bevor dies geschieht, werden die Ordner immer vollständig gelöscht.