KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Snel clusteren van tijdreeksen via lage-rang benaderingen

Mathias Pede

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotoren:

Prof. dr. R. Vandebril
Dr. ir. W. Meert

Assessoren:

Prof. dr. ir. D. Nuyens
ir. V. Vercruyssen

Begeleider:

ir. T. Steel

Academiejaar 2019 – 2020

Voorwoord

Deze thesis is met de hulp van meerdere mensen tot stand gekomen die ik dan ook van harte wens te bedanken hiervoor.

In de eerste plaats dank ik mijn promotoren Raf Vandebril en Wannes Meert om mij vanaf de eerste meeting de goede richting aan te geven en nadien nog geregeld bij te sturen. Dank ook aan mijn begeleider Thijs Steel, om altijd bereid en beschikbaar te zijn om in discussie te gaan over mijn onderwerp en mijn vragen te beantwoorden. Een paar onderdelen van deze thesis zijn gebaseerd op een aantal recente onderzoeken (minder dan 5 jaar oud), die soms zeer theoretisch waren en daarom niet altijd eenvoudig te begrijpen waren. Ik wil daarom mijn begeleider extra bedanken om samen met mij het hoofd te hebben gebroken over deze complexe papers en ze te helpen omzetten in de praktische toepassingen die deze thesis aanbiedt.

Dank tot slot aan mijn ouders voor de jarenlange motivatie en ondersteuning bij mijn studies.

Mathias Pede

Inhoudsopgave

Samenvatting

In de huidige informatiegedreven wereld met een exponentieel groeiende hoeveelheid data is er een toenemende behoefte aan technieken om data efficiënter te analyseren. Eén veelvoorkomend type data zijn *tijdreeksen*, waarbij sequentieel waarnemingen gedaan worden in de tijd.

Deze thesis focust zich op de ontwikkeling van algoritmes die de analyse via *clusteren* efficiënter maken. Bij clusteren worden n tijdreeksen ingedeeld in groepen met intern een grote gelijkenis. Deze gelijkenis tussen tijdreeksen wordt gedefinieerd door een afstandsfunctie. Om de clusters te creëren steunen klassieke clusteralgoritmes op een $(n \times n)$-matrix van alle paarsgewijze afstanden tussen alle tijdreeksen. Door de hoge dimensionaliteit van tijdreeksen is het evalueren van de afstandsfunctie tijdrovend en dus duur. Voor een grote dataset is het daarom niet haalbaar om deze volledige afstandsmatrix uit te rekenen en moet er gegrepen worden naar meer domeinspecifieke en gespecialiseerde clusteralgoritmes.

Dit werk tracht dit probleem op te lossen door de afstandsmatrix niet exact, maar benaderend te bepalen. Door de gelijkenissen tussen tijdreeksen is de afstandsmatrix in de praktijk goed te benaderen door een matrix van lagere rang k ($\ll n$). Om deze benadering op te stellen, volstaat het om slechts een fractie van de exacte afstandsmatrix te kennen. Dit werk bevat twee geïmplementeerde algoritmes, ACA en SOLRADM, beide gebaseerd op recent onderzoek, die een dergelijke benadering opstellen met slechts $\mathcal{O}(k)$ rijen van de matrix. Hiermee wordt het aantal evaluaties van een afstandsfunctie gereduceerd van $\mathcal{O}(n^2)$ naar $\mathcal{O}(k \cdot n)$.

In deze thesis wordt onderzocht en aangetoond dat beide algoritmes betere benaderingen vinden dan het state-of-the-art *framework* SPRL. In experimenten met 38 verschillende grote datasets werd aangetoond dat SOLRADM het berekenen van de afstandsmatrix $\mathcal{O}(n/k)$ keer sneller maakt en dat dit bovendien gebeurt zonder een significant verlies in kwaliteit van de gevonden clusters. Voor de grootste dataset van 24000 tijdreeksen betekende dit een versnelling van ongeveer factor 40 en deze factor neemt verder toe voor nog grotere datasets. De conclusie van dit onderzoek is dus dat het clusteren van grote datasets van tijdreeksen aanzienlijk efficiënter kan gemaakt worden via lage-rang benaderingen, vooral met SOLRADM.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen en symbolen

Afkortingen

ACA	Adaptive Cross Approximation
(A)RI	(Adjusted) Rand Index
DBA	DTW Barycentric Averaging
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DTW	Dynamic Time Warping
ED	Euclidean Distance (Euclidische afstand)
HAC	Hierchical Agglomerative Clustering
MSM	Move-Split-Merge
SOLRADM	Sample-Optimal Low Rank Approximation for Distance Matrices
SPRL	Similarity Preservering Representation Learning for Time Series Clustering
SVD	Singular Value Decomposition (Singulierewaardenontbinding)

Symbolen

η	Relatieve fout van de benadering
μ	Gemiddelde van een opmeting
σ	Standaardafwijking van een opmeting (Niet te verwarren met σ_i)
σ_i	De i-de grootste singuliere waarde (van een matrix)
$\times n$	Bemonsteringsfactor

Hoofdstuk 1

Introductie

Iedereen is vertrouwd met de uitspraak "Meten is weten". Technologische vooruitgang maakt het steeds eenvoudiger om data op te meten en ook te bewaren. De goedkopere prijs en hogere nauwkeurigheid van sensoren hebben ertoe geleid dat machines en gebouwen vandaag de dag volop uitgerust zijn met meetinstrumenten. Elk meetinstrument doet frequent opmetingen doorheen de tijd en vormt op deze manier een *tijdreeks*. Tijdreeksen hoeven niet noodzakelijk voort te komen uit opmetingen van sensoren, maar kunnen bijvoorbeeld ook de rubrieken uit de bedrijfsbalansen op het einde van opeenvolgende jaren zijn. Zo zijn tijdreeksen alomtegenwoordig in domeinen zoals geneeskunde, economie en wetenschap. Door de opkomst van *cloud storage* is geheugen praktisch gratis[1], en kunnen dergelijke tijdreeksen allemaal worden opgeslagen. Veel kennis kan vergaard worden uit de analyse van al deze meetgegevens. Zo is er een groeiende interesse naar analyses van de tijdreeksen voor patroonherkenning, classificatie, predictie en meer.

Eén van de populairste *machine learning*-technieken voor het analyseren van een verzameling van data is clusteren. Hierbij worden alle objecten ingedeeld in groepen, met intern een grote gelijkenis. Er bestaat een ruim aanbod aan technieken om deze clusters te bepalen[7]. Deze thesis richt zich op de clustertechnieken waarbij het algoritme vertrekt van een matrix met alle paarsgewijze afstanden tussen alle tijdreeksen, in praktijk vaak via de afstandsfunctie *Dynamic Time Warping (DTW)*. Voorbeelden hiervan zijn hiërarchisch clusteren, spectraal clusteren en DBSCAN. Deze technieken zijn populair omdat ze eenvoudig toe te passen zijn en bovendien weinig specifieke voorkennis van het domein van de data behoeven.

Het opstellen van de matrix met paarsgewijze afstanden voor een verzameling van n tijdreeksen heeft een tijdscomplexiteit van $\mathcal{O}(n^2 \cdot a)$ met a de tijdskost van één evaluatie van de afstandsfunctie. Deze kost a is afhankelijk van de lengte l van de tijdreeksen. Voor de klassieke versie van DTW is deze kost $\mathcal{O}(l^2)$, een hoge kost dus voor lange tijdreeksen. De complexiteit van het clusteren zelf, het effectief

[1] Een abonnement voor een team met een ongelimiteerde hoeveelheid opslagruimte op Dropbox kost in 2020 slechts 15 euro per maand per gebruiker[6].

opdelen in groepen, variëert van algoritme tot algoritme. Voorbeelden zijn $\mathcal{O}(n^2)$ voor DBSCAN[8] en $\mathcal{O}(n^3)$ voor spectraal clusteren[9]. Het berekenen van de afstandsmatrix heeft dus een significant aandeel in de totale complexiteit van het hele clusterproces van tijdreeksen en maakt het daarom ook slecht schaalbaar. Dit is het probleem dat deze thesis probeert op te lossen. In een verzameling tijdreeksen zijn er mogelijk verschillende tijdreeksen die erg op elkaar gelijken. De afstandsmatrix zal dus rijen of kolommen bevatten die ongeveer gelijk zijn aan elkaar. Of anders gezegd, de matrix kan mogelijk goed benaderd worden door een matrix van lagere rang. Er kan dan de vraag gesteld worden of het mogelijk is om een lage-rang benadering van de afstandsmatrix op te stellen zonder dat de gehele matrix gekend is. Door adaptief te kiezen welke matrixelementen geëvalueerd moeten worden en zo een lage-rang benadering op te stellen, kan mogelijks het aantal dure evaluaties van de afstandsfunctie worden gereduceerd.

Dit werk stelt twee algoritmes voor die hierin slagen en het aantal evaluaties reduceren naar $\mathcal{O}(k \cdot n)$, met k de rang van de benadering. Beide algoritmes zijn geïmplementeerd in een softwarebibliotheek[10]. Het eerste algoritme, ACA, voegt iteratief rang-één matrices toe aan de benadering tot een gewenste nauwkeurigheid bereikt is. Deze techniek is algemeen toepasbaar, maar is niet altijd in staat om een zeer nauwkeurige oplossing te vinden. Het tweede algoritme, SOLRADM, maakt bijkomend gebruik van de extra relaties tussen de elementen van de afstandsmatrix via de driehoeksongelijkheid, en biedt garanties op een hogere nauwkeurigheid van de benadering. De driehoeksongelijkheid geldt wel enkel voor een metrische afstandsfunctie, hetgeen DTW niet is. Onze experimenten tonen echter aan dat DTW doorgaans *bijna* voldoet aan de driehoeksongelijkheid en daarom nog steeds goed werkt in combinatie met SOLRADM. We vergelijken deze resultaten ook met twee afstandsfuncties die wel metrisch zijn. De Euclidische afstand (ED) is een klassieke afstandsfunctie, maar heeft weinig toepassingen door zijn beperkte flexibiliteit. Daarom gebruiken we ook Move-Split-Merge (MSM), een metrische afstandsfunctie die bovendien gelijkaardig aan DTW is. We testten SOLRADM op een divers aanbod van datasets en bereikten een gemiddelde relatieve fout van 0.02 voor rang-50 benaderingen. Voor de grootste dataset (24000 tijdreeksen) waren hiervoor slechts 2.38% van de matrixelementen nodig, circa 40 keer sneller dan de matrix volledig berekenen. Daarnaast tonen de experimenten aan dat de clusters via de benadering kwalitatief zeer dicht in de buurt komen van de clusters via de exacte afstandsmatrix.

In Hoofdstuk 2 wordt eerste achtergrondinformatie gegeven en worden notaties geïntroduceerd. Vervolgens worden in Hoofdstuk 3 een aantal gerelateerde werken besproken en wordt in Hoofdstuk 4 de probleemstelling toegelicht. De driehoeksongelijkheid en de afstandsfunctie MSM worden in detail besproken in Hoofdstuk 5. In dit hoofdstuk introduceren we ook een techniek om te kwantificeren in welke mate een matrix de driehoeksongelijkheid schendt. In Hoofdstuk 6 volgt de theoretische basis en de praktische implementatie van de algoritmes die de lage-rang benadering berekenen. Ten slotte worden de kwaliteit van deze algoritmes en de impact op de volledige clusterpijplijn onderzocht in verschillende experimenten in Hoofdstuk 7.

Hoofdstuk 2

Achtergrond

Dit hoofdstuk bespreekt de nodige achtergrond voor deze thesis. In Sectie 2.1 wordt een tijdreeks gedefiniëerd en kort de methodiek voor het vergelijken van tijdreeksen besproken. Vervolgens worden in Sectie 2.2 enkele clusteralgoritmes aangehaald en een evaluatietechniek geformuleerd om de kwaliteit van de gevonden clusters te beoordelen. Ten slotte bevat Sectie 2.3 de notaties en enkele eigenschappen in verband met matrices die in dit werk gebruikt worden.

2.1 Tijdreeks

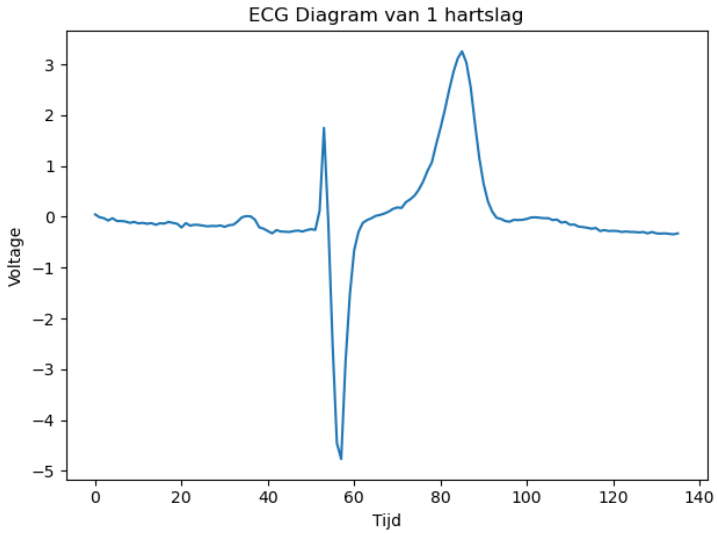
Definitie 1. Een tijdreeks T is een sequentie van l waarden $t_i \in D$ uit een domein D, geïndexeerd volgens hun tijdstip,

$$T = \{t_1, t_2, ..., t_{l-1}, t_l\}. \tag{2.1}$$

De tijdreeks is *univariaat* als het domein D slechts uit één variabele bestaat (bv. temperatuur). In het geval van meerdere variabelen praat men over een *multivariate* tijdreeks (bv. de snelheid volgens de x-, y- en z-as). Verder noteren we ook $T(i) = t_i$.

In dit werk beschouwen we enkel univariate tijdreeksen op het domein $\mathbb{R}$ en nemen we aan dat het tijdsinterval tussen meetwaarden constant is, waardoor elke meetwaarde een even grote bijdrage heeft over de tijd. Een voorbeeld van een tijdreeks is gegeven in Figuur 2.1. Wanneer er geen meetwaarde is voor een bepaald tijdstip, wordt de tijdreeks soms aangevuld met ontbrekende waarden $t_i = \texttt{null}$ om aan te geven dat er geen meetwaarde is voor dit tijdstip. Dergelijke tijdreeksen vallen buiten het bereik van deze thesis, maar kunnen wel omgezet worden naar een tijdreeks zonder ontbrekende waarden via technieken zoals interpolatie of predictie.

Tijdreeksen komen voor in verschillende domeinen, bv. economie en geneeskunde. In de economie zijn tijdreeksen belangrijk voor analyses van prijzen en bedrijfsresultaten [11]. Er bestaan talrijke strategieën voor technische analyse van aandelenprijzen [12, 13, 14]. Ook in de geneeskunde zijn tijdreeksen frequent aanwezig bij het opmeten



Figuur 2.1. Een voorbeeld van een univariate tijdreeks met lengte 136. De grafiek representeert het electrocardiogram(ECG) van één enkele hartslag. De grafiek werd opgesteld met data afkomstig van de *ECGFiveDays* dataset uit het UCR Time Series Archive[1].

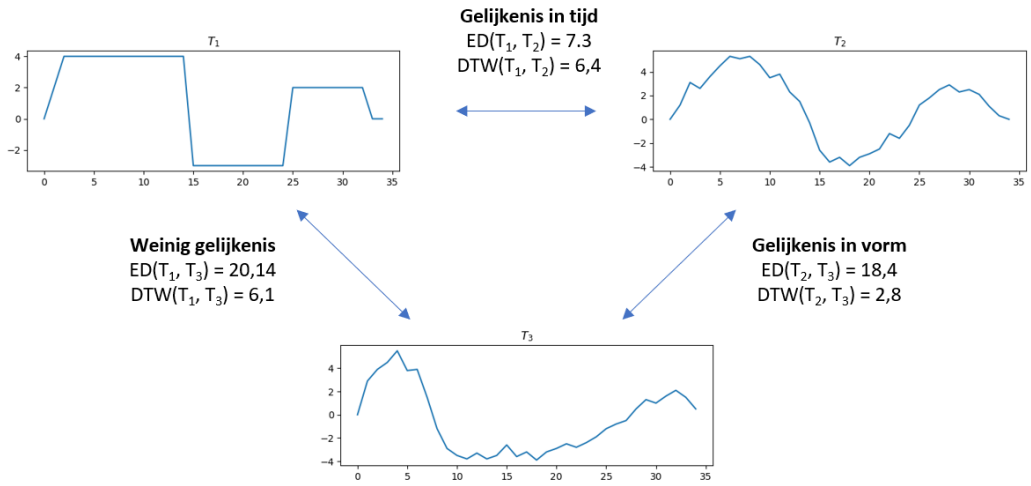
van hartslagen (Figuur 2.1), bloeddruk en andere metingen om de gezondheidstoestand van de patiënt te controleren, bv. [15].

Door de technologische vooruitgang op het vlak van netwerken (IoT) en geheugenopslag is het steeds eenvoudiger om sequentiële data te verzamelen en op te slaan. Deze toenemende beschikbaarheid van data, in de vorm van tijdreeksen, leidt op haar beurt tot een grote vraag naar tijdreeksanalyses om nuttige informatie uit de data te generen. Dit heeft ook als gevolg dat onderzoek naar betere en snellere analysemethodieken een zeer actueel onderwerp is.

Voor het uitvoeren van een clusteranalyse beschikken we over een verzameling $\mathcal{T} = \{T_1, ..., T_n\}$ van n tijdreeksen. In eerste instantie is het dan belangrijk om de gelijkenis of afstand tussen verschillende tijdreeksen te definiëren en te kwantificeren.

2.1.1 Tijdreeksen vergelijken

Om een verzameling tijdreeksen in te delen in groepen, is het nodig om tijdreeksen onderling te kunnen vergelijken. Dit is echter minder vanzelfsprekend dan het berekenen hoe ver twee coördinaten uit elkaar liggen. Namelijk, vanaf wanneer lijken twee tijdreeksen op elkaar en hoe valt dit te kwantificeren? Een voor de hand liggend antwoord bestaat niet. Twee tijdreeksen kunnen op elkaar gelijken wat betreft gemiddelde en variantie, maar toch een heel verschillend verloop hebben.



Figuur 2.2. Een voorbeeld van 3 tijdreeksen waarbij de notie van gelijkenis verschilt. T_1 en T_2 vertonen voornamelijk gelijkenis in tijd, omdat hun fluctuaties min of meer gelijktijdig verlopen. Dit vertaalt zich in een lage ED afstand (zie verder). T_2 en T_3 verlopen niet gelijktijdig, maar hun vormen lijken wel meer op elkaar (rondere pieken). Deze gelijkenis in vorm wordt beter gecapteerd door de DTW afstand (zie verder).

Omgekeerd kan ook: twee tijdreeksen kunnen hetzelfde verloop hebben, maar door een verschil in magnitude kunnen ze toch apart van elkaar geklasseerd worden. Als dit ongewenst is en de magnitude van de tijdreeksen geen rol speelt, moeten de tijdreeksen eerst genormaliseerd worden. Daarna herkent het werk [7] drie mogelijke vormen van gelijkenissen tussen tijdreeksen: gelijkenis in tijd, vorm en verandering. Deze thesis richt zich op de eerste twee. Als tijd de belangrijke factor is, bijvoorbeeld bij onderzoek naar slaapritmes, dan moet de afstandsfunctie ook de gelijkenis bepalen op basis van min of meer overeenkomstige tijdstippen. Andere afstandsfuncties laten meer verschuivingen in de tijd toe, waardoor het aspect van de vorm van de tijdreeks meer naar boven komt. Het verschil wordt geïllustreerd in Figuur 2.2.

Kortom, een data-analyst heeft meestal zelf nog wat vrijheid nodig om een goede afstandsfunctie te kiezen of op te stellen. Daarom testen we onze algoritmes in deze thesis met drie verschillende afstandsfuncties: ED, DTW en MSM. ED en DTW worden in de volgende secties besproken. MSM is complexer en bespreken we in Hoofdstuk 5.

Normalisatie

Vaak is een data-analyst niet geïnteresseerd in de absolute waardes van de tijdreeks. De data zijn mogelijk opgesteld met een verschillende schaal of misschien zijn de absolute waarden van weinig belang voor de analyse. Neem bijvoorbeeld een

verzameling tijdreeksen van de prijsevolutie van verschillende aandelen op de markt. Afhankelijk van bijvoorbeeld de bedrijfsactiviteit en het aantal uitstaande aandelen kunnen de prijzen van de aandelen grondig verschillen. Niettegenstaande kunnen sommige aandelen een erg gelijkaardige prijsevolutie hebben, bijvoorbeeld aandelen uit dezelfde sector. De absolute waarde is hier duidelijk niet van belang. In dat geval is het nuttig om de tijdreeksen eerst te normaliseren. Een eenvoudige manier is het herschalen van een tijdreeks T naar de tijdreeks N tussen $[-1, 1]$ via

$$N(i) = \frac{T(i)}{\max(T)}. \tag{2.2}$$

Omdat deze begrensde herschaling nogal gevoelig is voor potentiële uitschieters in de tijdreeks, wordt meestal geopteerd voor de z-normalisatie. Na z-normalisatie wordt een tijdreeks T omgezet naar een tijdreeks Z met

$$Z(i) = \frac{T(i) - \mu_T}{\sigma_T} \tag{2.3}$$

met μ_T het gemiddelde en σ_T de standaardafwijking van T.

Na normalisatie kunnen tijdreeksen dan in relatieve waarde met elkaar vergeleken worden. De meest bekende methoden hiervoor zijn de afstandsfuncties ED en DTW.

Euclidische afstand (ED)

Gemakshalve kan een tijdreeks gezien worden als een vector van lengte l. Hij kan dus geplaatst worden in de Euclidische ruimte $\mathbb{R}^l$, waarin een afstand gedefinieerd is via de p-norm. Omdat alle tijdreeksen, die vergeleken worden, deel moeten uitmaken van deze ruimte, is deze afstand opmeten enkel mogelijk wanneer alle tijdreeksen dezelfde lengte l hebben, of moet er een stuk van de langste tijdreeks afgeknipt worden. De onderlinge p-afstand tussen twee tijdreeksen T_1 en T_2 van lengte l is gedefiniëerd als

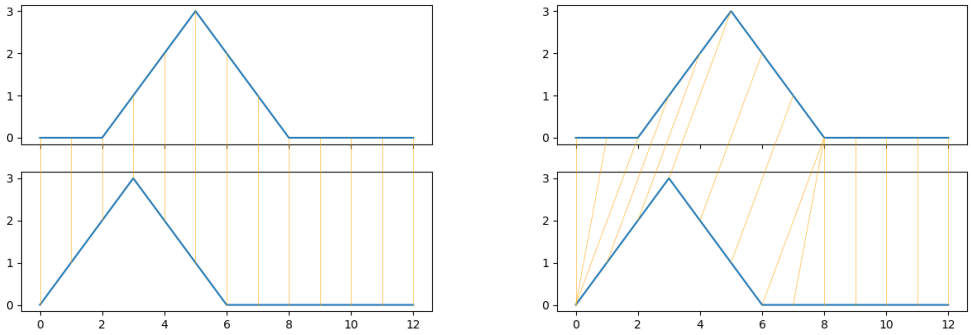
$$d_p(T_1, T_2) = \left(\sum_{i=1}^{l} |T_2(i) - T_1(i)|^p \right)^{1/p} \quad \text{voor } p \geq 1. \tag{2.4}$$

Indien $p = 2$, praat men over de Euclidische afstand(ED) en wordt (2.4) simpelweg de afstand tussen twee punten in een coördinatenstelsel. Andere waarden van p komen in de praktijk zelden voor. De berekening ervan is efficiënt in tijd en geheugen, nl. $\mathcal{O}(l)$, omdat enkel overeenkomstige tijdstappen met elkaar vergeleken worden, zoals te zien in Figuur 2.3a. ED neemt dus voornamelijk gelijkenis in tijd waar. Nadeel daarvan is dat deze *one-to-one* alignering dus geen vat heeft op eventuele temporele verschuivingen.

Dynamic Time Warping (DTW)

DTW lost het probleem van temporele verschuivingen op door een *many-to-many* alignering te berekenen tussen beide tijdreeksen. Dit is belangrijk omdat het mogelijk

is dat twee tijdreeksen dezelfde inhoud representeren, maar op een verschillende snelheid. DTW werd voor het eerst voorgesteld in de context van spraakherkenning in [16]. Bij spraakherkenning doet de snelheid er niet zo veel toe, maar moet vooral het patroon van de tijdreeks geanalyseerd worden om woorden te herkennen. Het DTW algoritme biedt hiervoor een oplossing door op zoek te gaan naar de optimale alignering, zodanig dat de afstand tussen de overeenkomstige punten minimaal is. Een voorbeeld is te zien in Figuur 2.3b. Bijkomend voordeel van de flexibele alignering is dat de tijdreeksen niet meer beperkt moeten worden tot dezelfde lengte. Een voorbeeld van de alignering en het verschil met ED is te zien in Figuur 2.3.



(a) Euclidische afstand **(b)** Dynamic Time Warping

Figuur 2.3. Het effect van temporele verschuivingen op de berekening van de afstand tussen beide tijdreeksen bij ED en DTW. ED heeft een *one-to-one* alignering en gaat dus een grotere afstand vinden dan DTW, waarbij de *many-to-many* alignering het toelaat de gelijkenissen in beide tijdreeksen te vinden. Opgesteld via de `dtaidistance`[2] softwarebibliotheek.

Het algoritme bestaat uit twee stappen: de kosten van alle mogelijke aligneringen berekenen en vervolgens op basis van deze kosten de optimale alignering selecteren. Voor twee gegeven tijdreeksen T_1 en T_2 met lengtes l en m, wordt er eerst een kost-matrix $C \in \mathbb{R}^{l \times m}$ opgesteld. Elk element $C(i, j)$ is de kost wanneer de eerste i tijdstippen van T_1 optimaal gealigneerd worden met de eerste j tijdstippen van T_2. De opeenvolgende elementen $C(i, j)$ kunnen berekend worden door te beginnen met $C(1, 1) = |T_2(1) - T_1(1)|$ en vervolgens de formule

$$C(i, j) = |T_2(j) - T_1(i)| + min[C(i-1, j), C(i, j-1), C(i-1, j-1)] \tag{2.5}$$

toe te passen. Deze methode moet gevolgd worden tot de hele matrix is ingevuld en dan heeft de optimale alignering een totale kost $C(l, m)$. Deze kost is de DTW-afstand.

Het opstellen van de kost-matrix is de duurste stap van DTW en heeft een complexiteit van $\mathcal{O}(l \cdot m)$. Als men enkel geïnteresseerd is in de getalwaarde van de gelijkenis, dan kan men na deze stap stoppen. Indien men ook wil weten hoe de

alignering precies gebeurt, dan wordt in de tweede stap via de kost-matrix het pad bepaald met de laagste kost. Dit kan gebeuren in lineaire tijd $\mathcal{O}(l+m)$ en dus blijft de totale complexiteit van DTW $\mathcal{O}(l \cdot m)$.

Deze kwadratische complexiteit is vaak een struikelblok bij het vergelijken van lange tijdreeksen. Een alternatieve versie van DTW is *constrained DTW* (cDTW), waarbij de maximale tijdsverschuiving w (= *warp venster*) in de alignering een parameter is van het algoritme. Dit venster bepaalt het maximaal aantal tijdstappen tussen twee gealigneerde tijdstippen. Formule (2.5) wordt dan enkel toegepast voor de elementen $C(i,j)$ met $|i-j| \leq w$. De complexiteit van cDTW wordt dan $\mathcal{O}(w \cdot max(l,m))$. De keuze van de grootte van dit venster is echter niet triviaal en hangt af van de onderliggende data. Er bestaan algoritmes om de optimale grootte van dit venster te berekenen, zoals [17]. Wanneer het venster 0 is, krijgen we simpelweg de Euclidische afstand.

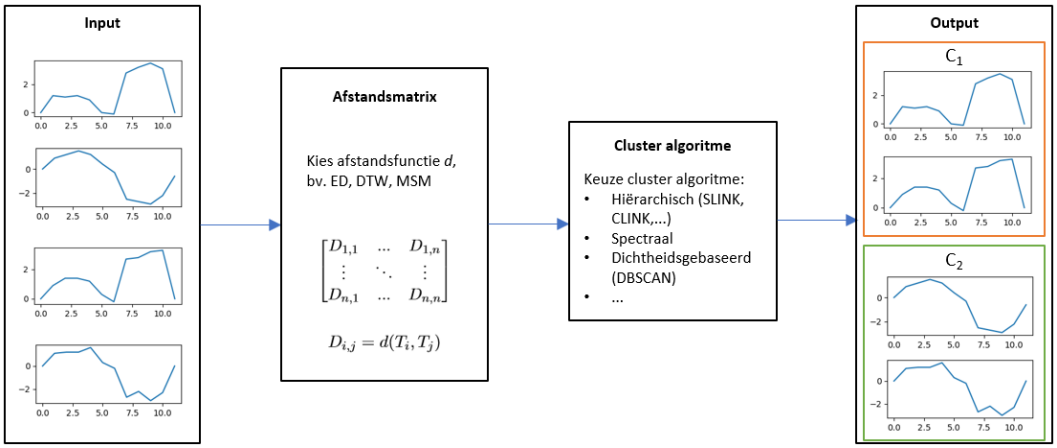
Een snellere, bijna-optimale versie van DTW bestaat in FastDTW[18], die reeds richting lineaire tijdscomplexiteit gaat. Maar voor erg lange tijdreeksen ($> 100\,000$) geldt deze lineaire tijdscomplexiteit niet meer. In deze thesis wordt gewone DTW zonder venster gebruikt.

Een belangrijke opmerking is dat er in de literatuur vaak verwezen wordt naar DTW als een *afstand*, maar in theorie voldoet DTW niet aan de axioma's van een afstand. In lijn met de literatuur en gebruiksgemak, zal verder in deze thesis DTW nog steeds een afstandsfunctie genoemd worden. De axioma's van een afstand worden nog verder besproken in Hoofdstuk 5 en we introduceren daar ook nog een derde afstandsfunctie, met name Move-Split-Merge, die wel aan deze axioma's voldoet.

2.2 Clusteren

Definitie 2. (Gebaseerd op [7]) Clusteren van tijdreeksen is het proces waarbij een gegeven dataset van n tijdreeksen $\mathcal{T} = \{T_1, ..., T_n\}$ gepartitioneerd wordt in $\mathcal{C} = \{C_1, ..., C_k\}$ zodanig dat, gebaseerd op een afstandsfunctie, tijdreeksen met onderling kleine afstanden samen gegroepeerd zitten. C_i wordt dan een *cluster* genoemd en er geldt $\mathcal{T} = \cup_{i=1}^{k} C_i$ en $C_i \cap C_j = \varnothing$ als $i \neq j$.

Dit werk richt zich op clusteralgoritmes waarbij de clusters bepaald worden op basis van alle paarsgewijze afstanden tussen de tijdreeksen van de dataset $\mathcal{T}$. Deze afstanden worden dan op voorhand berekend via een afstandsfunctie $d(\cdot,\cdot)$ (bv.



Figuur 2.4. Een overzicht van de gehele clusterpijplijn voor tijdreeksen. In de eerste stap moet de keuze gemaakt worden voor een afstandsfunctie, waarmee een afstandsmatrix opgesteld wordt. Vervolgens wordt deze matrix als input gebruikt voor een clusteralgoritme, dat ten slotte de berekende clusters als output heeft. Dit werk focust zich op het versnellen van de eerste stap.

DTW) en opgeslagen in een matrix $D \in \mathbb{R}^{n \times n}$ volgens

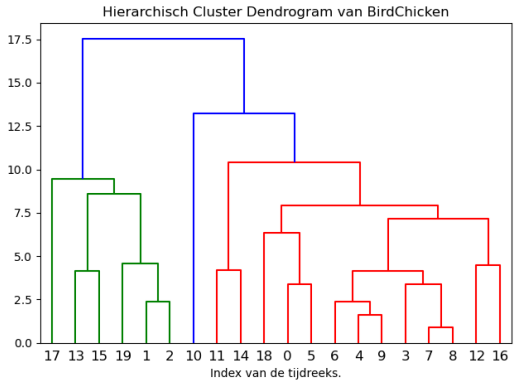
$$\begin{bmatrix} 0 & d(T_1,T_2) & \dots & d(T_1,T_{n-1}) & d(T_1,T_n) \\ d(T_2,T_1) & 0 & \dots & d(T_2,T_{n-1}) & d(T_2,T_n) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d(T_{n-1},T_1) & d(T_{n-1},T_2) & \dots & 0 & d(T_{n-1},T_n) \\ d(T_n,T_1) & d(T_n,T_2) & \dots & d(T_n,T_{n-1}) & 0 \end{bmatrix}. \tag{2.6}$$

Door $d(T_i, T_j) = d(T_j, T_i)$ (symmetrie-eigenschap van afstandsfuncties) is deze matrix symmetrisch, en moeten in totaal dus $n(n-1)/2$ elementen berekend worden. Deze matrix wordt als input gebruikt voor het clusteralgoritme om de tijdreeksen in groepen te delen. De clusteralgoritmes die verder in dit werk gebruikt worden zijn hiërarchisch, spectraal en dichtheidsgebaseerd clusteren. Een overzicht van de gehele clusterpijplijn voor tijdreeksen is te zien in Figuur 2.4.

2.2.1 Clusteralgoritmes

In dit werk wordt gebruik gemaakt van drie verschillende clusteralgoritmes, nl. hiërarchisch, spectraal en dichtheidsgebaseerd clusteren. Bij clusteralgoritmes is de kwaliteit van de output afhankelijk van het soort clusters in de tijdreeksen, maar ook van de gekozen afstandsfunctie. In de praktijk worden daarom vaak verschillende algoritmes toegepast op dezelfde data. Om deze reden richt dit werk zich ook op meerdere algoritmes om de algemeenheid te behouden, met name:

- **Hiërarchisch clusteren** [19]: Bij deze clustertechniek wordt een hiërarchie van alle te clusteren elementen gebouwd. Een voorbeeld is te zien in Figuur 2.5. De hiërarchie kan *bottom-up* of *top-down* opgebouwd worden. Bij



Figuur 2.5. Een voorbeeld van clusters berekend via een hiërarchie gebouwd met HAC. Deze hiërarchie is gebouwd met de trainingsdata van de *BirdChicken* dataset uit het UCR Time Series Archive[1]. Tijdreeksen worden samengevoegd in clusters op basis van *complete linkage* via hun DTW afstand. De hoogte waarop twee clusters samengevoegd worden, geeft aan hoe ver de clusters uit elkaar liggen. De drie kleuren stellen de drie gevonden clusters voor.

bottom-up worden steeds de dichtsbijzijnde clusters samengevoegd (*hierarchical agglomerative clustering* (HAC)). *Top-down* werkt omgekeerd en splitst telkens een cluster in twee clusters met de grootste afstand (*hierarchical divisive clustering* (HDC)). Door het verschil in tijdscomplexiteit ($\mathcal{O}(n^3)$ voor HAC vs $\mathcal{O}(2^n)$ voor HDC)) wordt meestal HAC gebruikt. Een voordeel van hiërarchisch clusteren is dat het aantal clusters niet gekend moet zijn op voorhand.

Initieel vormt elke tijdreeks zijn eigen cluster. Vervolgens worden de twee clusters samengevoegd met de kleinste onderlinge afstand. Deze afstand tussen twee clusters kan op verschillende manier gedefiniëerd worden en dit heeft ook invloed op de tijdscomplexiteit, nl.:

- *Single linkage*: De afstand tussen twee clusters wordt bepaald op basis van de afstand tussen de dichtstbijzijnde objecten uit beide clusters:

$$SL(C_1, C_2) = min\{d(x, y) : x \in C_1, y \in C_2\}. \tag{2.7}$$

- *Complete linkage*: De afstand tussen twee clusters wordt bepaald op basis van de afstand tussen de verst afgelegen objecten uit beide clusters:

$$CL(C_1, C_2) = max\{d(x, y) : x \in C_1, y \in C_2\}. \tag{2.8}$$

Een optimale implementatie (respectievelijk SLINK [20] en CLINK [21]) reduceert de $\mathcal{O}(n^3)$ tijdscomplexiteit in deze gevallen naar $\mathcal{O}(n^2)$.

- **Spectraal clusteren**[9]: Bij spectraal clusteren steunt het algoritme op de spectrale eigenschappen van een matrix. Het spectraal clusteralgoritme bestaat uit twee stappen. Het algoritme vertrekt van een gelijkenismatrix $G \in \mathbb{R}^{n \times n}$ (af te leiden uit de afstandsmatrix) en berekent de Laplaciaan

$$L = M - G, \tag{2.9}$$

met $M \in \mathbb{R}^{n \times n}$ een diagonaalmatrix met elementen $M_{i,i} = \sum_{j=1}^{n} G_{i,j}$. Van deze Laplaciaan L worden vervolgens de eerste k eigenvectoren $u_1, ..., u_k$ berekend. Elke waarde van een eigenvector kan geassocieerd worden met een te clusteren object en vormt zo een *feature vector* (bv. voor de eerste tijdreeks $[u_1(1), ..., u_k(1)]$). In de tweede stap wordt dan het klassieke clusteralgoritme *k-means* gebruikt om de clusters van alle objecten te bepalen. Er bestaan varianten waarbij de Laplaciaan in 2.9 anders berekend wordt. In het geval van een afstandsmatrix D, kan deze eerst omgezet worden in een gelijkenismatrix G via

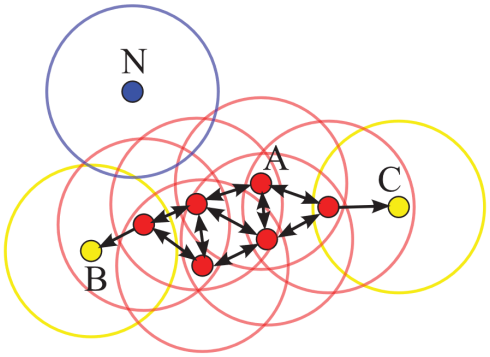
$$G_{i,j} = e^{-\gamma D_{i,j}^2}. \tag{2.10}$$

In dit werk stellen we $\gamma = max(D) - min(D)$. Voor het berekenen van de grootste eigenvectoren moet het SVD (zie Sectie 2.3.2) van de Laplaciaan opgesteld worden. Deze berekenen heeft een complexiteit van $\mathcal{O}(n^3)$ en is ook de duurste stap van spectraal clusteren.

- **Dichtheidsgebaseerd clusteren**: Density-Based Spatial Clustering of Applications with Noise (BDSCAN)[8] is een veelgebruikte techniek, die ook reeds in praktijk gebruikt is voor tijdreeksen in [22] en [23]. Het gebruikt rechtstreeks de afstanden uit de afstandsmatrix D om te bepalen of een object behoort tot een cluster of niet. Hiervoor introduceert DBSCAN het concept van *dichtheid-bereikbaar* zijn. Om dichtheid te definiëren zijn er twee parameters nodig: de maximale afstand ε en het minimum aantal objecten *minPts* dat binnen deze afstand ε moet liggen om een intern punt van een cluster te zijn. Een voorbeeld van hoe DBSCAN werkt is te zien op Figuur 2.6. Voor goede clusters is het nodig om verschillende parameterwaarden te testen en daarvoor is dus een beperkte kennis van de onderliggende data van de tijdreeksen nodig. Een voordeel is dat het aantal clusters niet gekend moet zijn op voorhand. DBSCAN heeft een tijdscomplexiteit van $\mathcal{O}(n^2)$.

2.2.2 Evaluatie clusters

Om empirisch de kwaliteit van een hele clusterpijplijn te bevestigen, moeten de gevormde clusters geëvalueerd worden. In dit werk evalueren we clusters op basis van de *Adjusted Rand Index*(ARI)[24], een verbeterde versie van de *Rand Index (RI)*[25]. Het RI criterium bekijkt elk mogelijk paar van tijdreeksen. Voor een partitie $\mathcal{C} = \{C_1, ..., C_k\}$ zitten beide tijdreeksen ofwel in dezelfde cluster, ofwel in een verschillende cluster. Twee partities $\mathcal{C}$ en $\mathcal{G}$ worden dan vergeleken op basis



Figuur 2.6. Een voorbeeld van hoe DBSCAN de clusters vormt. De maximale afstand ε is aangegeven door de cirkels rond de objecten en er moeten drie objecten binnen deze omgeving zijn om tot de interne punten (rood) van de cluster gerekend te worden. B en C zijn randobjecten en zijn bereikbaar vanaf de cluster, maar het is mogelijk dat een andere cluster wel drie objecten in de buurt heeft. N is een *ruisobject* en heeft geen andere objecten in de buurt. Afbeelding afkomstig uit [3].

van het aantal overeenkomsten dat ze hebben over het totaal aantal paren. Als a het aantal paren is dat beide partities in éénzelfde cluster plaatst, en b het aantal paren is dat beide partities in aparte clusters steken, dan wordt de RI-score voor een dataset met n tijdreeksen weergegeven als volgt:

$$RI(\mathcal{C}, \mathcal{G}) = \frac{a+b}{\frac{n(n+1)}{2}}. \tag{2.11}$$

Voor een volledig willekeurige partitie zullen er nog altijd overeenkomsten kunnen gevonden worden met een andere partitie. Daarom wordt in de ARI-score deze RI-score nog gecorrigeerd om ervoor te zorgen dat een willekeurige partitie een verwachtingswaarde van 0 heeft. Een grote gelijkenis tussen de partities geeft een ARI-score van 1 terug. Bij partities met weinig gelijkenis is de ARI-waarde laag, zelfs negatief als er minder overeenkomsten zijn dan verwacht.

2.3 Notatie en concepten matrix

2.3.1 Notatie

$A_{i,j}$ is het element in rij i en kolom j van de matrix A. De i-de rij van een matrix A wordt genoteerd als $A_{i,:}$ en de j-de kolom van een matrix wordt voorgesteld als $A_{:,j}$.

Definitie 3. $\|A\|_F$ is de Frobenius norm van de matrix $A \in \mathbb{R}^{m \times n}$ en wordt berekend via

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j}^2}. \tag{2.12}$$

Definitie 4. $\|A\|_2$ is de 2-norm van de matrix $A \in \mathbb{R}^{m\times n}$ en wordt berekend via

$$\|A\|_2 = \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sigma_{max}(A) \tag{2.13}$$

met $\sigma_{max}(A)$ de grootste singuliere waarde van A (zie Sectie 2.3.2).

Definitie 5. $\|A\|_C$ is de C-norm of ook wel de max-norm van matrix $A \in \mathbb{R}^{m\times n}$ en komt neer op het grootste element in absolute waarde van de matrix M:

$$\|A\|_C = \max_{i,j} A_{i,j}. \tag{2.14}$$

Met het volume $\mathcal{V}$ van een matrix A bedoelen we de $|det(A)|$. De kolomruimte van A is de ruimte opgespannen door de kolommen van A. De rijruimte van A is dan weer opgespannen door de rijen van A. Twee kolommen of rijen a_1 en a_2 zijn orthogonaal wanneer hun inwendig product $a_1 \cdot a_2$ gelijk is aan 0.

2.3.2 Singuliere waardenontbinding

Theorie 1. *Elke matrix $A \in \mathbb{R}^{m\times n}$ kan ontbonden worden in*

$$A = U\Sigma V^T \tag{2.15}$$

met $U \in \mathbb{R}^{m\times m}$ en $V \in \mathbb{R}^{n\times n}$ orthonormale matrices en $\Sigma \in \mathbb{R}^{m\times n}$ een pseudo-diagonaalmatrix (alle waarden buiten de diagonaal zijn 0). Deze ontbinding noemt men de singuliere waardenontbinding of SVD (Singular Value Decomposition) van de matrix en is uniek op permutaties van de rijen en kolommen na.

De kolomvectoren in $U = [u_1, ..., u_m]$ en $V = [v_1, ..., v_n]$ noemt men respectievelijk de linkse en rechtse singuliere vectoren. De elementen σ_i op de diagonaal van Σ zijn de singuliere waarden van de matrix en we nemen aan dat deze gerangschikt zijn van groot naar klein. Voor elke $i \leq \min(m, n)$ geldt dat $Av_i = \sigma_i u_i$.

Definitie 6. De rang r van een matrix $A \in \mathbb{R}^{m\times n}$ is het aantal lineair onafhankelijke kolommen van A. Deze rang is equivalent aan de dimensie van de kolomruimte van A. Als de matrix van volle rang is, is $r = \min(m, n)$.

Door vergelijking (2.15) verder uit te werken naar

$$A = U\Sigma V^T = \sum_{i=1}^{\min(m,n)} \sigma_i u_i v_i^T \tag{2.16}$$

kan de matrix ontbonden worden tot een som van kolom-rij producten (matrices van rang één). Omdat alle kolommen u_i orthogonaal zijn t.o.v. elkaar, is de dimensie van de kolomruimte dus gelijk aan het aantal rang-één matrices in de sommatie of het aantal singuliere waarden verschillend van 0. Deze decompositie toont ook de impact

van de singuliere waarden aan. De vectoren u_i en v_i zijn genormeerd, dus de normen van de rang-één matrices in (2.16) worden volledig bepaald door de grootte van de singuliere waarden. Het spreekt voor zich dat de rang-één matrix met de kleinste singuliere waarde dus de kleinste invloed heeft op de totale matrix A. Hieruit komt het idee van een lage-rang benadering, door de matrices behorende tot de kleinste singuliere waarden weg te laten vallen. Door slechts de k grootste singuliere waarden en hun bijhorende vectoren bij te houden wordt de rang-k benadering A_k bepaald via

$$A_k = U_k \Sigma_k V_k^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T. \tag{2.17}$$

Hierin bevat $\Sigma_k \in \mathbb{R}^{k \times k}$ de grootste k singuliere waarden en zitten hun bijhorende singuliere vectoren in $U_k \in \mathbb{R}^{m \times k}$ en $V_k \in \mathbb{R}^{n \times k}$. Deze benadering A_k is de oplossing van het minimalisatieprobleem

$$\underset{\tilde{A}, rang(\tilde{A}) \leq k}{\operatorname{argmin}} \; \|A - \tilde{A}\|_F \tag{2.18}$$

en heeft een relatieve fout

$$\eta_{best} = \frac{\|A - A_k\|_F}{\|A\|_F} = \frac{\sqrt{\sum_{i=k+1}^{n} \sigma_i^2}}{\sqrt{\sum_{i=1}^{n} \sigma_i^2}}. \tag{2.19}$$

De singuliere waarden van praktische matrices kunnen sterk aflopend zijn, waardoor vanaf een bepaalde k in vergelijking (2.19) de fout heel klein wordt. Dat geldt ook voor de matrices in deze thesis. In andere literatuur zoals [26] vermeldt men dit ook als ε-rang$(A) = k$ als er een foutmatrix E bestaat met $\|E\| = \mathcal{O}(\varepsilon)$ zodanig dat

$$rang(A + E) = k. \tag{2.20}$$

De singuliere vectoren van de matrices U_k en V_k vormen een orthonormale basis van respectievelijk de kolom- en rijruimte van benadering A_k, die een lagere dimensie heeft dan die van A. De lage-rang benadering uit (2.17) kan ook gezien worden als een projectie van de originele ruimte van A naar deze gereduceerde ruimte. De matrices $U_k U_k^T$ en $V_k V_k^T$ zijn dan projectiematrices naar deze kolom- en rijruimte, waarvoor geldt dat $A_k = U_k U_k^T A$ en $A_k = A V_k V_k^T$.

Voor de ontbinding 2.15 bestaan verschillende algoritmes, waarvoor de tijdscomplexiteit $\mathcal{O}(min(m^2 n, mn^2))$ is. Voor deze algoritmes is het echter nodig dat alle elementen van de matrix gekend zijn op voorhand. Een lage-rang benadering kan in dat geval nuttig zijn voor datacompressie, bv. Principal Component Analysis[27]. Voor het onderzoek in deze thesis zijn we echter op zoek naar een manier om een lage-rang matrix op te stellen zonder dat alle elementen van de matrix berekend moeten worden, om op deze manier een goede benadering te vinden van de niet-berekende matrix elementen.

Hoofdstuk 3

Gerelateerd werk

Dit hoofdstuk bevat enkele relevante en recente werken in verband met het clusteren van tijdreeksen. In Sectie 3.1 vermelden we een recent werk dat aan de basis ligt van de motivatie van deze thesis en waar een vergelijkbaar probleem wordt opgelost. De auteurs stellen een algoritme voor om een lage-rang benadering te berekenen voor een gelijkenismatrix van tijdreeksen om daarmee een *feature vector* voor elk tijdreeks te bepalen. In Sectie 3.2 vermelden we nog een andere clustertechniek waarbij het door het gebruik van *clusterprototypes* niet nodig is om de volledige afstandsmatrix te berekenen. Het opstellen van dergelijke prototypes is echter ingewikkeld, door de complexiteit van tijdreeksen. *DTW Barycentric Averaging* is een mogelijke techniek om deze prototypes op te stellen.

3.1 SPRL

De hoge kost van het opstellen van de matrix met paarsgewijze gelijkenissen van een verzameling tijdreeksen is ook de auteurs van [28] niet ontgaan. Zij stellen daarom het *Similarity Preserving Representation Learning (SPRL) framework* voor tijdreeksen voor. SPRL zet elke tijdreeks T_i om in een *feature vector* $[x_{i1}, ..., x_{ik}]$ van dimensie k. Verder clusteren gebeurt dan met deze *feature vectors* via *k-means*.

De omzetting van een tijdreeks naar een *feature vector* gebeurt op basis van de lage-rang benadering van een gelijkenismatrix G. In hun werk wordt een theoretisch bewijs geleverd dat wanneer de tijdreeksen behoren tot c clusters die goed te onderscheiden zijn, de matrix G een ε-rang heeft van maximaal $c(c-1)+2$ voor een kleine ε. Het framework vertrekt van een gelijkenismatrix G en stelt voor deze matrix een rang-k benadering $G \approx XX^T$ op, met $X \in \mathbb{R}^{n \times k}$. Elke rij van X is dan de nieuwe *feature vector* representatie van een tijdreeks. Deze techniek is enkel toepasbaar op een gelijkenismatrix, waardoor afstanden tussen tijdreeksen eerst omgezet moeten worden in een gelijkenis. Voor een verzameling tijdreeksen $\mathcal{T} = \{T_1, ..., T_n\}$ en een afstandsfunctie d (bv. DTW), wordt de gelijkenismatrix $G \in \mathbb{R}^{n \times n}$ bepaald door

$$G_{i,j} = \frac{d(T_i, 0)^2 + d(T_j, 0)^2 - d(T_i, T_j)^2}{2d(T_i, 0)d(T_j, 0)}, \tag{3.1}$$

met 0 de tijdreeks van lengte 1 met één enkele 0. De essentie van SPRL is dat deze matrix G niet helemaal berekend moet worden, maar dat de benadering berekend wordt op basis van slechts $\mathcal{O}(n \cdot log(n))$ elementen van G. Deze elementen worden volledig willekeurig gekozen en de matrix $X \in \mathbb{R}^{n \times k}$ is dan de oplossing van het minimalisatieprobleem

$$\min_X \|P_\Omega(G - XX^T)\|_F^2, \tag{3.2}$$

met P_Ω een projectie naar de op voorhand berekende waarden van G. Het *Efficient Exact Cyclic Coordinate Descent* algoritme[28] lost dit minimalisatieprobleem op in een klein aantal iteraties (± 10) met elke iteratie een tijdskost van $\mathcal{O}(k \cdot n \cdot log(n))$. Hoewel het werk geen resultaten geeft over de relatieve fout van XX^T ten opzichte van G, zijn de resultaten voor SPRL in combinatie met *k-means* wel veelbelovend.

Het gebruik van de omzetting naar de *feature vector* is niet echt relevant voor dit werk. Wel is het zo dat de benadering XX^T terug omgezet kan worden naar een afstandsmatrix door (3.1) om te keren naar $d(T_i, T_j)$ omdat alle andere termen exact gekend zijn. De benaderingsfout van de gelijkenismatrix zal daardoor vergelijkbaar zijn met de fout op de afstandsmatrix.

3.2 Clusteren met prototypes

Er bestaan clusteralgoritmes waarbij het niet nodig is om alle paarsgewijze afstanden tussen objecten te berekenen. In plaats van objecten onderling te vergelijken, wordt per cluster een *clusterprototype* berekend. Dit prototype representeert alle objecten in de cluster en is dus bij voorkeur een object dat goed gelijkt op de clusterobjecten. Bij het herindelen van de clusters worden de objecten dan vergeleken met de prototypes en niet met alle andere objecten. Als het aantal clusters k veel kleiner is dan het totaal aantal objecten n en daarnaast het berekenen van nieuwe prototypes snel en efficiënt kan gebeuren, dan kunnen deze clusteralgoritmes significant sneller werken. Het voornaamste voorbeeld van deze techniek is partitioneel clusteren, maar er zijn ook variaties op hiërarchisch clusteren die werken met cluster prototypes[29].

3.2.1 Partitioneel clusteren

Bij partitioneel clusteren worden de n objecten ingedeeld in k groepen of clusters. Het algoritme is zeer eenvoudig en is te zien in Algoritme 1. *K-means*, *k-medoids* en *k-centroids* zijn voorbeelden van implementaties van dit algoritme en verschillen op basis van hoe het clusterprototype bepaald wordt. Het algoritme kan beginnen met een willekeurige verdeling van de clusters of willekeurige objecten gebruiken als eerste k prototypes. Alle objecten worden dan opnieuw ingedeeld in clusters door te vergelijken met de prototypes. Bij elke iteratie wordt dan voor elke cluster een nieuw prototype opgesteld en de indeling opnieuw uitgevoerd. Voor het herindelen moeten dan enkel de $k \cdot n$ afstanden tussen alle n objecten en k prototypes van de clusters

berekend worden, met als gevolg een goede schaalbaarheid van het algoritme.

Algorithm 1: Partitioneel clusteren

Input: k (# clusters), $\mathcal{T} = \{T_1, ..., T_n\}$, afstandsfunctie d

```
Kies k tijdreeksen willekeurig als cluster prototypes P = {P_1, ..., P_k}
while Clusters veranderd do
    for T_i in T do
        for j = 1..k do
            Bereken d(P_j, T_i) (afstand tot cluster C_j)
        end
        Voeg T_i toe aan dichtsbijzijnde cluster
    end
    for j = 1..k do
        Bereken nieuw prototype P_j voor cluster C_j
    end
end
```

Result: Clusters $\mathcal{C} = \{C_1, ..., C_k\}$

De algoritmes *k-means*, *k-medoids* en *k-centroids* berekenen elk op hun eigen manier nieuwe clusterprototypes, maar voor tijdreeksen brengt dit vaak problemen met zich meer:

- Gemiddelde tijdreeks (*means*): De tijdreeksen in de huidige cluster C_j worden uitgemiddeld tot P_j volgens

$$P_j = \frac{1}{|C_j|} \sum_{T_i \in C_j} T_i. \tag{3.3}$$

  Bij gelijkenis in tijd (ED) is deze techniek toepasbaar, omdat de gemiddeldes genomen worden over elke tijdstip. Echter wanneer gelijkenis in vorm belangrijk (bv. DTW) is, zal *k-means* slechte resultaten opleveren. Door de temporele alignering van DTW is de gemiddelde tijdreeks niet altijd de tijdreeks die dicht bij alle tijdreeksen van de cluster ligt[4].

- Optimale prototype (*centroid*): Het optimale prototype is de tijdreeks P_j waarbij de som van de kwadratische afstanden tot elke tijdreeks in de cluster C_j minimaal is, nl.

$$P_j = \underset{X}{\operatorname{argmin}} \sum_{T_i \in C_j} d(X, T_j)^2. \tag{3.4}$$

  Deze oplossing is ook gekend als de *Steiner* sequentie. Voor ruimtes met een metrische afstandsfunctie zoals ED, is deze oplossing simpelweg het gemiddelde. Echter bij het gebruik van DTW wordt dit probleem zeer complex, omdat ook de lengte van de optimale tijdreeks een vrije parameter wordt. Voor een cluster met z tijdreeksen heeft het oplossen van (3.4) tijd en geheugen exponentieel in z nodig[4].

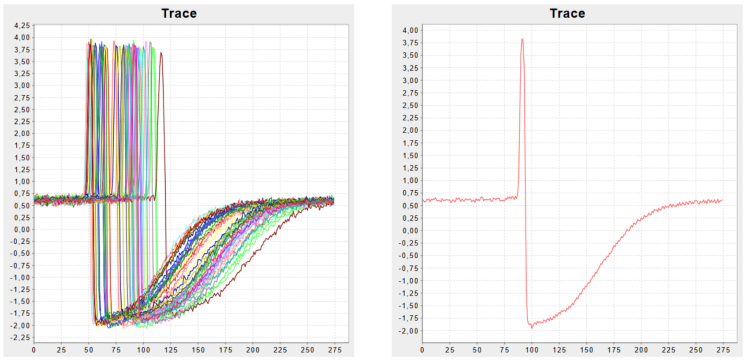
- *Medoid*: Omdat het zoeken naar het optimale prototype te moeilijk is, kan de zoekruimte verkleind worden tot de elementen in de cluster zelf. Dat wil zeggen dat er enkel gezocht wordt naar de tijdreeks P_j waarvoor geldt dat

$$P_j = \underset{T_x \in C_j}{\operatorname{argmin}} \sum_{T_i \in C_j} d(T_x, T_i)^2. \tag{3.5}$$

  Voor dit minimalisatieprobleem moeten nog altijd alle paarsgewijze DTW afstanden tussen alle tijdreeksen van een cluster berekend worden, waardoor het aantal gewonnen evaluaties van de afstandsfuncties beperkt blijft. Bovendien kunnen alle clusters nog veranderen tijdens het clusteralgoritme, waardoor de tijdswinst beperkt zal zijn of mogelijk zelfs negatief zal zijn door de extra op te lossen minimalisatieproblemen.

3.2.2 DTW Barycentric Averaging (DBA)

De populariteit van DTW heeft geleid tot nieuwe technieken om een vorm van gemiddelde tijdreeks voor een cluster te vinden, o.a. NLAAF[30] en PSA[31]. Beide algoritmes ondervinden echter problemen door de lengte van de tijdreeksen omdat ze zich baseren op het berekenen van lokale gemiddeldes. Een meer globale aanpak werd voorgesteld in [4] en [32] met *DTW Barycentric Averaging*(DBA). Dit algoritme probeert op een heuristische manier een oplossing te vinden op (3.4). Het algoritme begint met een initiële tijdreeks T_{DBA} als 'gemiddelde' en berekent de DTW alignering met elke tijdreeks in de cluster. Elke alignering zorgt voor een associatie van de waarden van T_{DBA} met de waarden van de tijdreeksen in de cluster (bv. $T_{DBA}(1)$ is geassocieerd met $T_1(1)$, $T_1(2)$, $T_2(1)$ voor een cluster met tijdreeksen T_1 en T_2). Van zodra al deze associaties gekend zijn, wordt het zwaartepunt van de associaties genomen als nieuwe meetwaarde in T_{DBA} ($(T_1(1) + T_1(2) + T_2(1))/3$ met de vorige associaties als voorbeeld). Dit proces kan dan een aantal keer herhaald worden tot er convergentie is. De geconvergeerde tijdreeks zou dan gebruikt kunnen worden als clusterprototype bij partitioneel clusteren.



Figuur 3.1. Een voorbeeld van een gemiddelde tijdreeks berekend via DBA (rechts) t.o.v. alle tijdreeksen in dezelfde cluster (links). Afbeelding afkomstig uit [4].

Hoofdstuk 4

Probleemstelling

Klassieke clusteralgoritmes zoals hiërarchisch, spectraal en dichtheidsgebaseerd clusteren delen een verzameling objecten op in clusters op basis van alle paarsgewijze afstanden. Deze paarsgewijze afstanden worden opgeslagen in een matrix D en behoren tot de invoer van het clusteralgoritme. Tijdreeksen zijn hoogdimensionele objecten, waardoor het uitvoeren van een afstandsfunctie d op twee tijdreeksen een dure operatie is. In het beste geval schaalt de tijdskost lineair met de lengte l van de tijdreeks (Sectie 2.1.1). Het berekenen van de afstandsmatrix D voor een verzameling $\mathcal{T} = \{T_1, ..., T_n\}$ met tijdreeksen van lengte l heeft dus een tijdskost van $\mathcal{O}(n^2 \cdot poly(l))$. Voor een grote verzameling met lange tijdreeksen zijn deze clusteralgoritmes dus moeilijk toepasbaar.

Men kan het berekenen van de volledige afstandsmatrix vermijden door de tijdreeksen eerst om te zetten naar een *feature vector*, een laagdimensionele representatie. Deze omzetting vereist echter goede domeinkennis van de datasets, die niet altijd aanwezig is. Een andere techniek is om DBA (Sectie 3.2.2) te gebruiken in combinatie met een partitioneel clusteralgoritme zoals *k-means*. Hierbij moeten echter een aantal parameters, zoals het aantal clusters c, gekozen worden. De analyse moet dan voor een aantal van deze parameters uitgevoerd worden, waardoor het nut van het vermijden van extra afstandsevaluaties afneemt. Bovendien is het altijd beter om verschillende clusteralgoritmes te gebruiken, omdat niet elk clusteralgoritme geschikt is voor het domein dat men onderzoekt, het type clusters, etc. Deze problemen tonen aan dat het nuttig is om te beschikken over de volledige afstandsmatrix D met alle paarsgewijze afstanden tussen alle tijdreeksen.

Onderzoeksvraag

Hieruit volgt de vraag of het mogelijk is om de matrix D te benaderen zonder alle elementen ervan exact te moeten berekenen en toch gelijkaardige clusters te verkijgen als bij de exacte matrix D. Door SPRL (Sectie 3.1) was het al duidelijk dat het mogelijk is om met $\mathcal{O}(n \cdot log(n))$ elementen een goede benadering van de gelijkenismatrix (afleidbaar uit de afstandsmatrix D) te vinden. Hun aanpak is echter naïef

en kiest de elementen van de matrix volledig willekeurig. We kunnen ons dan ook afvragen of het aantal matrixelementen nog meer gedrukt kan worden door gericht op zoek te gaan naar die elementen die het nuttigst zijn om een goede benadering op te stellen. Daarnaast biedt het werk van SPRL enkel empirische resultaten op de volledige cluster pijplijn en zegt het werk niets over de nauwkeurigheid van de benadering zelf.

Dit werk tracht op deze vraag een antwoord te bieden door grondig te kijken naar de structuur van deze matrix D en op basis daarvan stelt Hoofdstuk 6 twee algoritmes voor die een lage-rang benadering opstellen van D met adaptief-gekozen rijen en kolommen. Deze algoritmes steunen op de eigenschappen van afstandsfuncties die structuur geven aan de matrix D. Eén van deze eigenschappen is de driehoeksongelijkheid. Omdat de populaire afstandsfunctie DTW niet beschikt over deze eigenschap, maakt dit werk ook gebruik van Move-Split-Merge, een functie gelijkaardig aan DTW met driehoeksongelijkheid. Deze functie wordt in detail uitgelegd in Hoofdstuk 5. In datzelfde hoofdstuk zoeken we ook naar een manier om kwantitatief te bepalen hoe een DTW matrix afwijkt van een matrix met driehoeksongelijkheid. Ten slotte voeren we experimenten uit met onze algoritmes en probeert dit werk in Hoofdstuk 7 een antwoord te geven op het tweede deel van de onderzoeksvraag, namelijk hoe de fout van de benadering op de invoermatrix invloed heeft op de kwaliteit van de clusters, berekend via een clusteralgoritme.

Dit werk biedt ook concreet een softwarebibliotheek[10] aan met een implementatie van de algoritmes. Deze bibliotheek kan dan gebruikt worden om de afstandsmatrix snel te benaderen en zo het hele clusterproces te versnellen.

Hoofdstuk 5

Structuur van de afstandsmatrix

Met SPRL (Sectie 3.1) werd aangetoond dat de afstandsmatrix van objecten uit verschillende clusters goed benaderd kan worden door een lage-rang benadering. In dit hoofdstuk kijken we naar de eigenschap van de driehoeksongelijkheid. Deze eigenschap is enkel aanwezig wanneer de afstandsmatrix opgesteld is met een metrische afstandsfunctie, gedefinieerd in Sectie 5.1. Vermits de populaire afstandsfunctie DTW niet voldoet aan deze eigenschap, experimenteert dit werk ook met de metrische afstandsfunctie *Move-Split-Merge*, uitgelegd in Sectie 5.2, die wel voldoet aan de driehoeksongelijkheid. Ten slotte stelt dit werk in Sectie 5.3 een methodiek voor om te bepalen hoe ver een willekeurige matrix afwijkt van een matrix met een door driehoeksongelijkheid gebonden matrix.

5.1 Afstand

Zoals vermeld in Sectie 2.1.1 voldoet DTW niet aan de voorwaarden van een afstand. Dit heeft als gevolg dat de ruimte met de tijdreeksen die DTW als functie gebruikt om tijdreeksen te onderscheiden van elkaar, minder structuur vertoont dan een ruimte waarin een echte afstand gedefinieerd is.

Definitie 7. Een functie $d \in (V \times V \mapsto \mathbb{R})$ met V een verzameling van elementen, is een *afstand* of *metriek* als deze voldoet aan vier axioma's met $X, Y, Z \in V$:

$$\begin{aligned} &d(X,Y) \geq 0 \text{ (niet-negativiteit)}, \\ &d(X,Y) = 0 \iff X = Y \text{ (scheidingseigenschap)}, \\ &d(X,Y) = d(Y,X) \text{ (symmetrie)}, \\ &d(X,Z) \leq d(X,Y) + d(Y,Z) \text{ (driehoeksongelijkheid)}. \end{aligned} \tag{5.1}$$

ED voldoet aan alle axioma's en is dus een metrische afstandsfunctie. DTW daarentegen voldoet niet aan twee van deze axioma's. In de eerste plaats geldt de scheidingseigenschap niet. In Figuur 2.3 hebben de twee tijdreeksen door de perfecte mapping een DTW-afstand gelijk aan 0. Daarnaast voldoet DTW ook niet aan de driehoeksongelijkheid. Dit probleem is al herhaaldelijk aangehaald in literatuur zoals

[33, 34, 35]. Zonder de driehoeksongelijkheid zal de afstandsmatrix D telkens een symmetrische matrix zijn met nullen op de diagonaal zonder extra garanties op een interne structuur. In praktijk zal DTW de driehoeksongelijkheid echter meestal niet in grote mate schenden, m.a.w. de parameter b in

$$DTW(X,Z) \leq DTW(X,Y) + DTW(Y,Z) + b \tag{5.2}$$

om de ongelijkheid te doen opgaan, blijft klein. In Sectie 5.3 stellen we een algoritme voor om dit kwantitief uit te drukken over een gehele matrix. In de volgende sectie stellen we een afstandsfunctie voor, die wel voldoet aan alle eigenschappen van (5.1) en bovendien gelijkaardig is aan DTW (gelijkenis in vorm).

5.2 Move-Split-Merge (MSM)

De voordelen van een functie die wel voldoet aan de axioma's van een afstand, mogen niet onderschat worden. Een verzameling van elementen waarin een metrische afstand gedefinieerd is, noemt men een metrische ruimte. Een metrische ruimte heeft meer structuur dan een ruimte zonder afstandsfunctie (een topologische ruimte). Met deze motivatie ontwikkelden A. Stefan et al. *Move-Split-Merge* (MSM)[5]. Deze afstandsfunctie voor tijdreeksen moest voldoen aan de axioma's (5.1). Het resultaat van hun onderzoek is een algoritme dat gebaseerd is op Edit distance with Real Penalty (ERP)[36]. Twee tijdreeksen T_1 en T_2 worden met elkaar vergeleken door T_1 te transformeren naar T_2 met de basis-operaties *Move, Split en Merge*. Elke operatie heeft een bepaalde kost en de afstand tussen beide tijdreeksen is gedefinieerd door de minimale kost om de transformatie uit te voeren. Gegeven een tijdreeks $T = (t_1, ..., t_i, ..., t_n)$, zijn de operaties in [5] gedefinieerd als:

- **Move**: Voeg een getal p toe aan de i-de waarde in de tijdreeks T. De kost van deze operatie is de waarde van het toegevoegde getal, nl.:

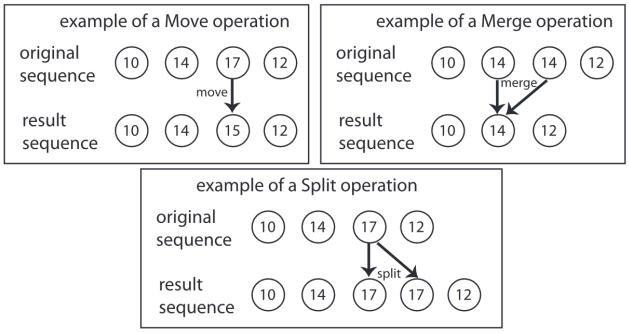
$$\begin{aligned} Move(T,i,p) &= \{t_1, ..., t_i + p, ..., t_n\}, \\ Kost(Move(T,i,p)) &= |p|. \end{aligned} \tag{5.3}$$

- **Split**: Splits (kopie) de i-de waarde van de tijdreeks. Het resultaat is dat de tijdreeks één element langer wordt. De kost van deze operatie is een constante $c > 0$, een parameter van MSM:

$$\begin{aligned} Split(T,i) &= \{t_1, ..., t_i, t_i, ..., t_n\}, \\ Kost(Split(T,i)) &= c. \end{aligned} \tag{5.4}$$

- **Merge**: Verwijder de waarde op plaats $i+1$ van de tijdreeks op voorwaarde dat $t_i = t_{i+1}$. Het resultaat is dat de tijdreeks één element kleiner wordt. De kost van deze operatie is equivalent aan de kost van een Split:

$$\begin{aligned} Split(T,i) &= \{t_1, ..., t_i, t_i + 2, ..., t_n\}, \\ Kost(Merge(T,i)) &= c. \end{aligned} \tag{5.5}$$



Figuur 5.1. Een voorbeeld met elk van de operaties Move, Split en Merge toegepast op een eenvoudige tijdreeks. Afbeelding afkomstig uit [5].

Een visuele voorstelling van deze operaties is te zien in Figuur 5.1. De drie operaties zijn voldoende om elke mogelijke tijdreeks X te transformeren in elke andere tijdreeks Y. De kosten van *Merge* en *Split* moeten verplicht hetzelfde zijn om te voldoen aan de symmetrie eigenschap van (5.1)[5]. Aan de driehoeksongelijkheid is altijd voldaan. Stel dat voor drie tijdreeksen X, Y en Z geldt dat $MSM(X,Z) > MSM(X,Y) + MSM(Y,Z)$, dan is de transformatie van $MSM(X,Z)$ niet de optimale transformatie van X naar Z. Immers transformeren van X naar Y, en vervolgens Y naar Z heeft een lagere kost. Hieruit volgt dat $MSM(X,Y) + MSM(Y,Z)$ een bovengrens is voor $MSM(X,Z)$, dus het voorafgaande is onmogelijk. MSM is reeds gecombineerd met *k-nearest-neighbors* en dit leverde veelbelovende resultaten voor verschillende classificatieproblemen[5].

De drie bouwblokken werden in [5] gebruikt om Algoritme 2 op te stellen. Net als bij DTW is er initieel een kost-matrix C, die sequentieel wordt ingevuld. Ook hier is de finale kost van de optimale transformatie dan te vinden op $C(m,n)$. De berekening van afzonderlijke elementen van de matrix is gelijkaardig aan DTW, maar is iets complexer door het gebruik van een hulpfunctie F. Deze functie F is in [5] gedefinieerd volgens

$$F(a,b,c) = \begin{cases} c & \text{if } b \leq a \leq c \text{ or } b \geq a \geq c \\ c + min(|a-b|, |a-c|) & \text{anders.} \end{cases} \tag{5.6}$$

Door het opstellen van de kost-matrix heeft Algoritme 2 net als DTW zonder venster een tijds- en geheugencomplexiteit van $\mathcal{O}(m \times n)$. Een venster toepassen op MSM is geen mogelijkheid, omdat dan de eigenschap van de driehoeksongelijkheid niet meer gegarandeerd kan worden. Deze thesis biedt in de softwarebibliotheek[10] een `Python` en een `C` versie van MSM aan. Daarnaast bevat het ook de mogelijkheid om parallel een gehele MSM matrix te berekenen.

Algorithm 2: Move-Split-Merge (MSM), uit [5]

Input: $X = (x_1, ..., x_m), Y = (y_1, ..., x_n)$
$C = 0^{m \times n} (\in \mathbb{R}^{m \times n})$
$C(1,1) = |x_1 - y_1|;$
for i = *2,...,m* **do**
  $C(i,1) = C(i-1,1) + F(x_i, x_{i-1}, y_1)$
end
for j = *2,...,n* **do**
  $C(1,j) = C(1,j-1) + F(y_i, x_1, y_{j-1})$
end
for i = *2,...,m* **do**
  for j = *2,...,n* **do**
    $C(i,j) = min\{C(i-1,j-1) + |x_i - y_j|,$
    $C(i-1,j) + F(x_i, x_{i-1}, y_j),$
    $C(i,j-1) + F(y_j, x_i, y_{j-1})\}$
  end
end
Result: $C(m,n)$

5.3 Driehoeksongelijkheid

Door de keuze van de afstandsfunctie, voldoen de elementen van de afstandsmatrix D wel of niet aan de driehoeksongelijkheid. Zoals eerder aangegeven geeft deze eigenschap meer structuur aan de matrix en dit laat ons toe deze structuur te gebruiken om de grootte van elementen in te schatten. Een voorbeeld ter illustratie is de situatie waarbij we beschikken over drie tijdreeksen X, Y en Z. Stel dat de afstanden $d(X,Y)$ en $d(Y,Z)$ gekend zijn en $d(X,Z)$ ongekend is. Als de afstand berekend is via een metrische afstand, en dus de driehoeksongelijkheid geldt, weten we dat $d(X,Y) + d(Y,Z)$ de bovengrens is voor $d(X,Z)$. Meer intuïtief zegt de driehoeksongelijkheid dat wanneer een object X dicht bij een object Y ligt, en object Y ligt erg dicht bij een object Z, dan volgt daaruit dat objecten X en Z ook dicht bij elkaar liggen. Deze inschatting kan niet gemaakt worden met DTW, maar wel met ED en MSM. Dankzij de driehoeksongelijkheid is het zelfs mogelijk om met één rij van de matrix, de normen van alle rijen in te schatten. Dit legt de basis van het tweede benaderingsalgoritme in het volgende hoofdstuk in Sectie 6.3.

5.3.1 Afstand tot een metrische afstandsmatrix

In de meeste gevallen zullen de afstanden berekend via DTW wel voldoen aan de driehoeksongelijkheid, maar DTW biedt geen garantie dat dit altijd het geval is. Een afstandsmatrix D opgesteld via DTW kan bijgevolg drie tijdreeksen T_x, T_y en T_z bevatten, waarvoor de driehoeksongelijkheid geschonden wordt met een marge $b > 0$

zodat

$$D_{x,z} = D_{x,y} + D_{y,z} + b. \tag{5.7}$$

Het aantal triplets van indices van D dat de eigenschap schenden als in (5.7) hangt af van de dataset. Niet alleen het aantal schendingen, maar ook de grootte b van de schending is van belang. Wanneer alle b's in het algemeen klein zijn voor de matrix D, dan zullen de inschattingen gemaakt op basis van de driehoeksongelijkheid slechts beperkt fout zijn. Indien b gekend is voor een gegeven triplet van indices, is het mogelijk (5.7) aan te passen volgens

$$(D_{x,z} - \frac{b}{3}) = (D_{x,y} + \frac{b}{3}) + (D_{y,z} + \frac{b}{3}) \tag{5.8}$$

door de fout b evenwaardig te verdelen over de drie elementen. Door ook effectief deze elementen in de matrix te corrigeren, voldoet dit triplet nu wel aan de driehoeksongelijkheid. Het is echter mogelijk dat deze correctie een nieuwe schending van de driehoeksongelijkheid introduceert in een ander triplet. Voor de afstandsmatrix D zijn we dus vooral geïnteresseerd in welke foutmatrix E we erbij moeten tellen zodanig dat het geheel een metrische afstandsmatrix is, met E zo klein mogelijk. Concreet zijn we dus op zoek naar de dichtsbijzijnde matrix X uit de verzameling van alle metrische afstandsmatrices ($\mathcal{M}_D$). Wiskundig komt dit neer op het minimalisatieprobleem

$$\arg\min_{X \in \mathcal{M}_D} \|X - D\| \tag{5.9}$$

Een algoritme om dit probleem op te lossen in de $\|\cdot\|_2$-norm werd voorgesteld in [37]. In dit algoritme worden iteratief alle mogelijke triplets van indices overlopen. In het geval van een schending van de ongelijkheid, worden de onderlinge afstanden gecorrigeerd om de grootte van de schending teniet te doen zoals in (5.8). Zoals vermeld kan dit schendingen introduceren in andere triplets, waardoor dit proces een aantal keer herhaald moet worden tot er convergentie is tot een metrische afstandsmatrix. In dit werk stellen we een aanpassing van dit algoritme voor in Algoritme 3. Het enige verschil met het algoritme van [37] is dat het stopcriterium anders werkt. Bij het itereren over alle triplets wordt er steeds bijgehouden hoeveel schendingen van de driehoeksongelijkheid er zijn. De correcties zijn geconvergeerd op het moment dat er geen schendingen meer zijn. Dit criterium is eenvoudiger te berekenen dan het voorgestelde criterium in [37] en leidt tot gelijkaardige resultaten.

Algoritme 3 is computationeel zwaar. Het aantal triplets dat getest moet worden voor een matrix $D \in \mathbb{R}^{n \times n}$ is $\mathcal{O}(n^3)$. Het convergeren naar de optimale oplossing gebeurt wel in een klein aantal iteraties, met een tijdscomplexiteit van $\mathcal{O}(n^3)$ per iteratie. Omdat de correcties van alle mogelijke triplets van vorige iteraties bijgehouden moeten worden, is ook het geheugengebruik $\mathcal{O}(n^3)$. De softwarebibliotheek van deze thesis bevat een implementatie van dit algoritme, gemaakt in `Python` en `C`. Door de geheugencomplexiteit is het nodig om het onderzoek te beperken tot middelgrote matrices (ca. 1500×1500). Grotere matrices vereisen veel RAM-geheugen en ook

Algorithm 3: *Triangle inequality fixing*, gebaseerd op [37]

Input: $D \in \mathbb{R}^{n \times n}$, kleine ε
for $0 \leq i < j < k < n$ **do**
  $(z_{ijk}, z_{jki}, z_{kij}) \leftarrow 0$
end
for $0 \leq i < j < n$ **do**
  $E_{ij} \leftarrow 0$
end
schendingen $\leftarrow n$
while *(schendingen* > 0*)* **do**
  schendingen $\leftarrow 0$
  for *triplet*(i, j, k) **do**
    $b \leftarrow D_{ki} + D_{jk} - D_{ij}$
    $\mu \leftarrow \frac{1}{3}(E_{ij} - E_{jk} - E_{ki} - b)$
    if $(\mu > \varepsilon)$ **then**
      schendingen $\leftarrow$ *schendingen* $+ 1$
      $\theta \leftarrow \min(-\mu, z_{ijk})$
      $E_{ij} \leftarrow E_{ij} + \theta, E_{ij} \leftarrow E_{jk} - \theta, E_{ki} \leftarrow E_{ki} - \theta$
      $z_{ijk} \leftarrow z_{ijk} - \theta$
    end
  end
end
Result: $X = D + E$ met $\arg\min_{X \in \mathcal{M}_D} \|X - D\|_2$

een enorm lange *runtime* door de hoge tijdscomplexiteit.

Voor een willekeurige afstandsmatrix D kan Algoritme 3 gebruikt worden om de matrix X te berekenen die voldoet aan (5.9). Omdat X de dichtsbijzijnde matrix uit $\mathcal{M}_D$ is, is het mogelijk om te kwantificeren hoe ver D verwijderd is van de ruimte van metrische matrices $\mathcal{M}_D$. De relatieve afstand tot de metrische ruimte definiëren we dan als

$$\frac{\|X - D\|_F}{\|D\|_F}. \tag{5.10}$$

met X de oplossing van (5.9), berekend via Algoritme 3. Deze afstand is nuttig voor de experimenten in Hoofdstuk 7. Nu kunnen we in het volgende hoofdstuk overgaan naar de introductie van onze algoritmes voor de lage-rang benadering.

Hoofdstuk 6

Lage-rang benadering

In Sectie 2.3.2 werd het concept van SVD reeds uitgelegd om de beste lage-rang benadering van een matrix op te stellen. In dit hoofdstuk stellen we twee algoritmes voor die een lage-rang benadering bepalen van een matrix zonder dat daarvoor alle elementen gekend moeten zijn. In Sectie 6.1 leggen we eerst het concept van een skelet decompositie uit en tonen daarmee aan dat het mogelijk is om een goede benadering te vinden met een beperkte selectie van kolommen en rijen. Vervolgens stelt dit werk in Sectie 6.2 een algoritme voor dat steunt op dit principe om een lage-rang benadering op te stellen. In Sectie 6.3 wordt ten slotte nog een tweede algoritme voorgesteld dat een andere techniek gebruikt om rijen en kolommen te selecteren en bovendien ook garanties geeft op de nauwkeurigheid van de resulterende benadering.

6.1 Skelet decompositie

Voor elke matrix $A \in \mathbb{R}^{m \times n}$ met rang r is het mogelijk om r rijen en r kolommen te selecteren en daarmee A exact te reconstrueren. Deze rijen en kolommen vormen vervolgens 'een skelet' van de matrix.

Definitie 8. (Gebaseerd op [38]) Gegeven een matrix $A \in \mathbb{R}^{m \times n}$ van rang r, dan bestaan er verzamelingen van indices

$$\hat{I} = \{i_1, ..., i_r\} \text{ voor de } r \text{ gekozen rijen en} \tag{6.1}$$

$$\hat{J} = \{j_1, ..., j_r\} \text{ voor de } r \text{ gekozen kolommen} \tag{6.2}$$

zodanig dat de matrices

$$R = A_{\hat{I},:}, \tag{6.3}$$

$$C = A_{:,\hat{J}} \text{ en} \tag{6.4}$$

$$\hat{A} = A_{\hat{I},\hat{J}} \tag{6.5}$$

de matrix A exact reconstrueren via

$$A = C\hat{A}^{-1}R. \tag{6.6}$$

Vergelijking 6.6 noemt men een *skelet decompositie* van de matrix A.

$$\begin{pmatrix} 2 & -2 & -11 & 7 \\ 6 & 4 & -3 & 5 \\ 8 & 2 & -14 & 12 \\ 2 & 3 & 4 & -1 \\ 10 & 5 & -10 & 11 \end{pmatrix} = \begin{pmatrix} 2 & -11 \\ 6 & -3 \\ 8 & -14 \\ 2 & 4 \\ 10 & -10 \end{pmatrix} \begin{pmatrix} 8 & -14 \\ 10 & -10 \end{pmatrix}^{-1} \begin{pmatrix} 8 & 2 & -14 & 12 \\ 10 & 5 & -10 & 11 \end{pmatrix}$$

Figuur 6.1. Een voorbeeld van een skelet decompositie van een matrix. De 5×4 matrix in het linkerlid heeft een rang van 2. Bijgevolg bestaat er een keuze van 2 rijen en kolommen zodanig dat vergelijking 6.6 opgaat. Deze keuze is niet uniek. Een andere combinatie van rijen en kolommen is ook een mogelijke skelet decompositie.

Een voorbeeld van een skelet decompositie is gegeven in Figuur 6.1. In tegenstelling tot het voorbeeld hebben matrices in de praktijk geen exacte lage rang r, maar een ε-rang r (Sectie 2.3.2). In dat geval zal (6.6) slechts bij benadering juist zijn en praat men over een *pseudoskelet*[38]. Dit pseudoskelet vormt dan een lage-rang benadering van de matrix A. Indien de matrix A goed benaderd kan worden door een matrix van rang r, dan bestaat er ook een pseudoskelet decompositie met een goede nauwkeurigheid[38]. Dit toont aan dat het potentieel mogelijk is om met weinig rijen en kolommen toch een matrix te benaderen.

De nauwkeurigheid van de benadering is echter afhankelijk van de gekozen rijen en kolommen. Deze rijen en kolommen bepalen de submatrix $\hat{A} = A_{\hat{I},\hat{J}}$. Het is bewezen in [38] dat er, indien $\hat{A}$ niet-singulier is, geldt dat

$$\|A - C\hat{A}^{-1}R\|_2^2 = \mathcal{O}(\|A\|_2^2 \cdot \|\hat{A}^{-1}\|_2^2 \cdot \varepsilon). \tag{6.7}$$

Uit de factor $\|\hat{A}^{-1}\|_2^2$ volgt dat de rijen en kolommen zodanig gekozen moeten worden dat $\hat{A}$ een zo laag mogelijk begrensde inverse heeft. Dit vertaalt zich in de zoektocht naar een submatrix $\hat{A}$ met een zo groot mogelijke volume[38, 39, 40, 41]. Het volume voor een matrix $B \in \mathbb{R}^{m \times m}$ is gedefinieerd als

$$\mathcal{V}(B) = |det(B)| = |\prod_{i}^{m} \sigma_i(B)|. \tag{6.8}$$

De rijen en kolommen kiezen met een submatrix met maximaal volume is echter geen eenvoudige zaak, zelfs een NP-hard probleem[42]. Daarnaast is het ook vaak mogelijk dat $\hat{A}$ slecht geconditioneerd is, waardoor het berekenen van de inverse leidt tot grote fouten. Daarom wordt $\hat{A}^{-1}$ soms vervangen door een andere matrix G[38]. In andere literatuur zoals [43] wordt deze matrix soms U genoemd, en dan spreekt men over een CUR decompositie. Het werk [40] bewijst verder nog een bovengrens voor de fout van de decompositie CGR met r rijen en kolommen (rang r) van

$$\|A - CGR\|_C \leq (r+1)\sigma_{r+1}(A) \tag{6.9}$$

met $\sigma_{r+1}(A)$ de $(r+1)$-de singuliere waarde van A. De $\|.\|_C$-norm (gedefinieerd in (2.14)) en de factor $(r+1)$ maken deze bovengrens relatief zwak. In het recentere werk [44] werd bewezen dat door het aantal rijen en kolommen te verhogen naar $(2r-1)$ en $G = \hat{A}_r^+$ (beste rang-r benadering van de pseudo-inverse van $\hat{A} \in \mathbb{R}^{(2r-1)\times(2r-1)}$) te nemen, de bovengrens verlaagd kan worden naar

$$\|A - C\hat{A}_r^+ R\|_C \leq 2\sigma_{r+1}(A). \tag{6.10}$$

Om te voldoen aan deze bovengrens moeten de rijen en kolommen gekozen worden zodanig dat het *r-geprojecteerde volume* van $\hat{A}$ maximaal is. Hierin is het r-geprojecteerde volume voor een matrix $B \in \mathbb{R}^{m\times m}$ gedefinieerd als

$$\mathcal{V}_r(B) = |\prod_i^r \sigma_i(B)|. \tag{6.11}$$

In tegenstelling tot (6.8) is het r-geprojecteerde volume dus enkel gebaseerd op de grootste r singuliere waarden. Daarnaast wordt in [44] ook bewezen dat wanneer r-geprojecteerde volume van $\hat{A}_r^+$ in (6.10) niet maximaal is, maar α keer kleiner is dan het maximum, dan zal de benadering ook hoogstens α keer slechter zijn. Het is dus duidelijk dat het ook met suboptimale rijen en kolommen mogelijk is om een goede benadering op te stellen.

Deze theorieën suggereren dus dat goede rang-k benaderingen gevonden kunnen worden met weinig rijen en kolommen als σ_{k+1} klein is. Algoritmes die in staat zijn om op een snelle manier kolommen en rijen te kiezen, die het maximaal volume goed benaderen, kunnen dus een goede benadering van de matrix vinden, zonder de hele matrix te moeten kennen. De algoritmes in de volgende secties hebben elk hun eigen manier om deze rijen en kolommen te kiezen.

6.2 Adaptive Cross Approximation

6.2.1 Theorie

De vorige sectie introduceerde het concept skelet decompositie. Nu is het ook mogelijk om een skelet decompositie van een matrix A te maken met slechts één rij en één kolom ($r = 1$). De rij en kolom kunnen gekozen worden door een willekeurig element $A_{i,j}$ van de matrix te kiezen. Visueel vormen deze rij en kolom een kruis (*cross*). Samen vormen ze een rang-één skelet decompositie

$$A \approx \frac{A_{i,:}A_{:,j}}{A_{i,j}} \tag{6.12}$$

met als residu

$$R \approx A - \frac{A_{i,:}A_{:,j}}{A_{i,j}}. \tag{6.13}$$

Door hetzelfde proces te herhalen op het residu wordt een steeds nauwkeurigere benadering van A gevormd. Deze techniek wordt *cross approximation* genoemd.

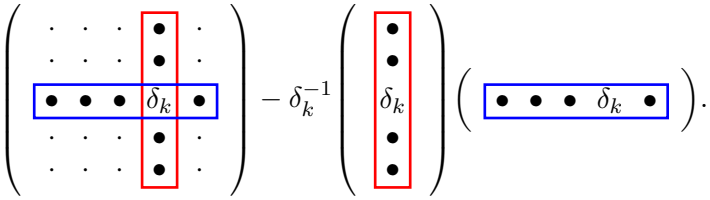
Het concept van *cross approximation* uitbreiden met een adaptieve keuze van de rij en kolom heeft geleid tot het algoritme *Adaptive Cross Approximation*(ACA)[26]. In de vorige sectie werd duidelijk gemaakt dat de rijen en kolommen van de skelet decompositie gekozen moeten worden zodat de gevormde submatrix een maximaal volume heeft. Doordat de decompositie telkens van rang één is bij *cross approximation* reduceert dit probleem zich tot het zoeken van het grootste element van de matrix. Omdat hiervoor alle elementen van de matrix gekend moeten zijn, zoekt ACA het grootste element in een bepaalde rij of kolom. ACA vertrekt van een willekeurige rij $w_1^{(r)} = A_{i,:}$ en vindt het grootste element δ_1 in absolute waarde op index j in deze rij. Daarom selecteert het de kolom $w_1^{(c)} = A_{:,j}$ en vormt zo een eerste rang-één benadering

$$A_1 \approx M_1 = \delta_1^{-1}(w_1^{(c)})(w_1^{(r)})^T \tag{6.14}$$

In de volgende stap wordt dezelfde methodiek herhaald, maar nu op de matrix $R_1 = A - M_1$. Het algoritme vertrekt deze keer vanaf de (niet eerder gebruikte) rij met het grootste element in de vorige kolom $w_1^{(c)}$. Bij elke iteratie wordt er dus een residu-matrix R_k berekend, door de vorige matrix R_{k-1} te benaderen via

$$R_k = R_{k-1} - \delta_k^{-1}(w_k^{(c)})(w_k^{(r)})^T \tag{6.15}$$

of visueel

$R_k =$  $- \delta_k^{-1}$.

Deze matrix moet zelf niet helemaal uitgerekend worden, enkel de rij en kolom van het volgende kruis. Op basis van $w_{k-1}^{(c)}$ wordt i_k^* gekozen als index van de volgende rij $w_k^{(r)}$ die dan berekend wordt via

$$w_k^{(r)} = A_{i_k^*,:} - \sum_{j=1}^{k-1} \delta_j^{-1}(w_j^{(c)})_{i_k^*}(w_j^{(r)})^T. \tag{6.16}$$

De kolom van het kruis wordt analoog berekend. Dit proces blijft zich herhalen en vormt zo iteratief de rang-k benadering

$$A \approx M_k = \sum_{i}^{k} \delta_i^{-1}(w_i^{(c)})(w_i^{(r)})^T. \tag{6.17}$$

6.2.2 Implementatie

In de oorspronkelijke versie van ACA uit [26] stopt het algoritme nadat een maximale rang k bereikt is. Een andere aanpak is kijken naar de laatst gekozen rij $w_i^{(r)}$ en stoppen van zodra $\|w_i^{(r)}\|_2$ klein is of het grootste element van de rij δ_k kleiner is dan een gekozen waarde[45]. In dit werk stellen we een eigen versie voor van ACA die steunt op de versie in [46], maar met een ander stopcriterium voorgesteld in [47], waarbij gebruik wordt gemaakt van een stochastische benadering van de norm. De afstandsmatrix D in dit werk is telkens een symmetrische indefiniete matrix, dus wensen we als benadering ook een symmetrische matrix. De symmetrische versie van [46] is hiervoor het vertrekpunt van onze versie van ACA. Om de symmetrie te behouden moeten de rij en kolom in 6.17 hetzelfde genomen worden en wordt de benadering dus

$$M_k = \sum_{i}^{k} \delta_i^{-1}(w_i)(w_i)^T. \tag{6.18}$$

In [46] wordt naast de huidige selectie van rijen ook nog een selectie van t willekeurige elementen van de matrix bijgehouden. Bij elke iteratie worden deze elementen vergeleken met hun benadering en het algoritme stopt van zodra de fout van alle benaderingen kleiner is dan een bepaalde waarde. Daarnaast kunnen deze elementen ook gebruikt worden voor de selectie van de rij voor het volgende kruis. Het algoritme van deze thesis neemt dat laatste idee over, maar het stopcriterium pakken we anders aan. In dit werk hechten we belang aan de relatieve fout van de benadering in de $\|\cdot\|_F$-norm. Daarom gebruiken we de t willekeurige monsters om een benadering van de $\|\cdot\|_F$-norm van R_k te berekenen. Voor de matrix $D \in \mathbb{R}^{n\times n}$ geldt immers dat

$$\|D\|_F = n\sqrt{\mu} \tag{6.19}$$

met μ de gemiddelde waarde van het kwadraat van alle elementen in de matrix. Deze norm kan stochastisch benaderd worden door willekeurig elementen uit de matrix te nemen. Het aantal elementen dat nodig is om de norm te benaderen binnen een gewenste foutmarge Δ en met een gewenste waarschijnlijkheid $(1-\alpha)$ kan berekend worden via de student t-verdeling. Concreet moeten er N willekeurige elementen $D_{i,j}^2$ uit D geselecteerd worden totdat een gemiddelde μ en standaardafwijking σ bereikt is, waarvoor geldt dat

$$\frac{t(\frac{\alpha}{2}, N-1)\sigma}{\sqrt{N}\mu} < \Delta \tag{6.20}$$

met t de waarde van de student t-verdeling[47]. Door de grootte van de matrices in dit werk, nemen we aan dat er minstens 100 elementen gekozen moeten worden. Met een zekerheid van 99% ($\alpha = 1\%$) geeft dit een t-waarde van $3,39$. Deze N elementen worden dan gebruikt om initieel de norm van D te benaderen en vervolgens bij elke iteratie de norm van R_k in te schatten, tot

$$\frac{\|R_k\|_{F,schatting}}{\|D\|_{F,schatting}} < \tau \tag{6.21}$$

geldt, met de τ de gewenste maximale relatieve fout van de benadering. Omdat een fractie van de N elementen ook gebruikt wordt om een nieuwe rij te kiezen en bijgevolg deze elementen een grotere kans hebben om goed benaderd te zijn, is de benadering $\|R_k\|_{F,schatting}$ meestal in kleine mate een onderschatting van de echte norm. Dit probleem minimaliseren we door de matrix gestratificeerd te bemonsteren. In plaats van volledig willekeurig de indexen van de elementen te kiezen, kiezen we een element uit elke rij. Op deze manier heeft elke rij min of meer hetzelfde aandeel in alle monsters. Onze experimenten in Sectie 7.2.3 en de resultaten in tabel B.2 en tabel B.1 tonen aan deze inschatting meestal goed in de buurt ligt van de echte fout. Het bemonsteringalgoritme is te zien in Algoritme 4. Bij elke iteratie worden er zo n monsters toegevoegd, één monster in elke rij. Het aantal iteraties is afhankelijk van de afstanden in de matrix zelf en de opgegeven foutmarge. In deze thesis gebruiken we 0.01 als foutmarge, die meestal bereikt wordt in een klein aantal iteraties.

Algorithm 4: Gestratificeerde bemonstering met $99\%(1-\alpha)$ zekerheid

Input: Onberekende $D \in \mathbb{R}^{n\times n}(n > 100)$, Foutmarge Δ

```
1  t = 3.39
2  S = {}
3  N = 0
4  while tσ/(√N μ) > Δ do
5  |   for i = 1..n do
6  |   |   Voor willekeurige j bereken D_{i,j} en voeg toe aan S
7  |   end
8  |   μ = Gemiddelde van {s^2 : s ∈ S}
9  |   σ = Standaarafwijking van {s^2 : s ∈ S}
10 |   N = N + n
11 end
```

Result: S met N monsters uit D

Algoritme 5 toont dan onze volledige versie van ACA. Deze is geïmplementeerd in de softwarebibliotheek[10]. Bij elke iteratie in lijnen 7-12 wordt een nieuw kruis toegevoegd aan de totale benadering. Per iteratie moet er zo één rij van de matrix D berekend worden, m.a.w. $\mathcal{O}(n)$ evaluaties van de afstandsfunctie. De test om te zien of δ_k gelijk is aan 0 op lijn 10 is belangrijk voor afstandsmatrices, waarbij de diagonaal uit nullen bestaat. Bijvoorbeeld in de allereerste iteratie is δ_1 altijd gelijk aan 0, waardoor δ_1^{-1} niet te berekenen valt. Daarom vervangen we δ_k door het maximum van w_k, zodat de elementen van het toegevoegde kruis begrensd zijn door $\max(w_k)$. Vervolgens wordt in lijnen 13-16 wordt voor elk monster uit S bepaald wat de huidige residu-waarde is in R_k. Met deze waarden wordt dan de norm van R_k ingeschat. Als aan (6.21) voldaan is, stopt het algoritme(lijn 23). Is dat niet het geval, dan wordt in lijnen 25-28 een nieuwe index van een rij bepaald om de iteratie opnieuw uit te voeren.

In het beste geval zal Algoritme 5 stoppen wanneer de getolereerde fout τ bereikt

Algorithm 5: Adaptive Cross Approximation (ACA) voor symmetrische matrices met relatieve fout op Frobenius norm als stop criterium

Input: Onberekende $D \in \mathbb{R}^{n \times n}$, Tolerantie τ, k_{max}, Δ_k

$k = 1$, $k_{best} = 1$
$i_0 = random(\{1, ..., n\})$
$Z = \{i_0\}$, $I = \{1, .., n\}$
Bereken S via Algoritme 4 met $\Delta = 0.01$
$D_{norm} = \sqrt{Gemiddelde(S)}$, $LowestNorm = \infty$
while $k \leq k_{max}$ **do**
    Bereken $D_{i_k,:}$
    $w_k = D_{i_k,:} - \sum_{j=1}^{k} \delta_j^{-1} (w_j)_{i_k} (w_j)^T$
    $\delta_k = (w_k)_{i_k}$
    if $\delta_k == 0$ **then**
        $\delta_k = max(w_k)$
    end
    for $s \in S$ **do**
        Stel r en c indices van s in D
        $s = s - \delta_j (w_k)_r (w_k)_c^T$
    end
    $R_{norm} = \sqrt{Gemiddelde(S)}$
    if $R_{norm} < LowestNorm$ **then**
        $LowestNorm = R_{norm}$
        $k_{best} = k$
    end
    if $(\frac{R_{norm}}{D_{norm}} < \tau)$ OR $(k_{best} + \Delta_k < k)$ **then**
        break
    end
    $i_{row} = \underset{x \notin Z}{\text{argmax}} |(w_k)_x|$
    $i_{samples} = \underset{x}{\text{argmax}} \{|s_{x,y}| : s \in S, x \notin Z, y \notin Z\}$
    Neem i_{k+1} de index van het grootste tussen $|(w_k)_{i_{row}}|$ en $|s_{i_{samples},y}|$
    $Z = Z \cup \{i_{k+1}\}$
    $k = k + 1$;
end

Result: $D_{approx} = \sum_{j=1}^{k_{best}} (w_j)(w_j)^T$

wordt. Als deze fout niet bereikt is na k_{max} iteraties, dan zal het algoritme stoppen en de benadering met de laagste geschatte fout teruggegeven, gevonden na k_{best} iteraties. Een andere aanpak die we verder in de experimenten ook toepassen is het algoritme te laten stoppen van zodra voor een groot aantal iteraties Δ_k (bv. 100) geen betere benadering gevonden werd.

Het algoritme biedt echter geen garanties dat het in staat is om een oplossing te vinden met relatieve fout τ. Daarnaast hebben de volgorde van de gekozen rijen ook invloed op het feit of er nog een beter resultaat gevonden kan worden. Het algoritme in de volgende sectie, kan wel sterke garanties bieden, maar doet daarvoor wel aan *overbemonstering*. ACA gebruikt elke bemonsterde rij voor een nieuwe rang één benadering en zolang het aantal monsters uit Algoritme 4 klein is, is het aantal evaluaties van een afstandsfunctie met kost a dus $\mathcal{O}(n \cdot k)$, met k de rang van de benadering, wat de totale tijdscomplexiteit op $\mathcal{O}(n \cdot k \cdot a)$ brengen.

6.3 Sample-Optimal Low-Rank Approximation for Distance Matrices (SOLRADM)

De techniek van ACA voor symmetrische matrices steunt enkel op de lage ε-rang van een matrix. We weten echter dat een afstandsmatrix D door de driehoeksongelijkheid nog meer structuur heeft, hetzij exact (ED en MSM), hetzij benaderd (DTW). Het werk [48] stelde reeds een (in tijd-)sublineaire (beter dan $\mathcal{O}(n^2)$) methode voor het benaderen van metrische afstandsmatrices voor. De meest recente verbetering op dit werk is bereikt in *Sample-Optimal Low-Rank Approximation for Distance Matrices (SOLRADM)*[49]. Het beschrijft een algoritme dat erin slaagt een goede benadering te vinden in tijd $\mathcal{O}(n \cdot poly(k, 1/\varepsilon))$ met k de rang van de benadering en ε een maatstaf voor de maximale fout. Maar vooral, het algoritme heeft hiervoor slechts $\mathcal{O}(k/\varepsilon)$ rijen (of kolommen want D is symmetrisch) van de matrix nodig en de auteurs bewijzen theoretisch dat dit de ondergrens is. Door de recentheid van het onderzoek (2019), is dit algoritme nog in weinig andere werken opgenomen. De softwarebibliotheek[10] van deze thesis bevat voorzover geweten de eerste publieke implementatie van SOLRADM, specifiek voor het gebruik bij afstandsmatrices van tijdreeksen.

Het algoritme bestaat uit drie stappen en is gegeven in Algoritme 6. Het algoritme is niet deterministisch en heeft als output twee matrices $U \in \mathbb{R}^{n \times k}$ en $V \in \mathbb{R}^{k \times n}$, waarvan het product met een hoge waarschijnlijkheid voldoet aan

$$\|D - UV\|_F^2 \leq \|D - D_k\|_F^2 + \varepsilon \|D\|_F^2, \tag{6.22}$$

bewezen in [49]. De benadering UV heeft dus hoogstens een additieve fout $\varepsilon\|D\|_F^2$ ten opzichte van de beste rang k benadering D_k. Onze experimenten in Sectie 7.2.3 tonen echter aan dat de gevonden benaderingen ver onder de bovengrens van 6.22 zitten. Deze ε heeft invloed op het aantal rijen van D die geëvalueerd moeten worden. De hoogste efficiëntie (in relatieve fout t.o.v. aantal rijen) van Algoritme 6 bekwamen we in onze experimenten voor $\varepsilon = 2.0$.

De theoretische basis en praktische algoritmes van de drie onderdelen van SOLRADM worden uitgelegd in de volgende secties. In de eerste stap wordt een kansverdeling opgesteld over alle kolommen van D, hetgeen theoretisch enkel mogelijk

Algorithm 6: SOLRADM voor symmetrische afstandsmatrices[49]

Input: Afstandsmatrix $D \in \mathbb{R}^{n \times n}$, ε-parameter
1 Bereken kansverdeling kolommen via Algoritme 8 (eventueel >1 keer)
2 Bereken U via Algoritme 7
3 Bereken V via Algoritme 9
Result: $U \in \mathbb{R}^{n \times k}$ en $V \in \mathbb{R}^{k \times n}$, met UV als benadering van D

is voor een metrische afstandsmatrix. De kansverdeling wordt in de tweede stap gebruikt om via de *Monte-Carlo methode* kolommen te selecteren en de top k linkse singuliere vectoren van D te benaderen, die een matrix U vormen. In de laatste stap wordt deze matrix U gebruikt om een regressieprobleem op te lossen om V te bepalen. In de volgende sectie beginnen we met het uitleggen van de tweede stap, omdat deze voorwaarden oplegt voor de kansverdeling van de eerste stap.

6.3.1 Monte-Carlo methode (Stap 2)

SOLRADM steunt op de resultaten van het werk van A. Frieze en R. Kannan in [50]. Daarin wordt het idee van de *Monte-Carlo methode* toegepast op een matrix om een lage-rang benadering van de matrix te vinden. Monte-Carlo integratie is een bekende toepassing van de methode waarbij een integraal benaderd wordt door de functie willekeurig te bemonsteren binnen het integratie-interval en dan het gemiddelde te nemen. Hetzelfde principe kan gebruikt worden bij een matrix D door willekeurig kolommen te kiezen. Deze kolommen bepalen een matrix S. De ruimte opgespannen door de top k linkse singuliere vectoren van S zal bij benadering gelijk zijn aan de ruimte opgespannen door de eerste k linkse singuliere vectoren van D. Een lage-rang benadering van D kan dan berekend worden door een projectiematrix (zie Sectie 2.3.2) met deze singuliere vectoren op te stellen.

Voor deze techniek moet a priori een kansverdeling $(p_1, p_2, ..., p_n)$ gekend zijn voor de willekeurige keuze van de kolommen van de matrix $D \in \mathbb{R}^{n \times n}$. Deze verdeling moet een redelijke benadering zijn van de echte norm van de kolommen. Meer specifiek moet er een constante c bestaan, met $0 < c \leq 1$, waarvoor geldt dat

$$p_i \geq c \frac{\|D_{:,i}\|_2^2}{\|D\|_F^2} \tag{6.23}$$

voor alle $i \in \{1, .., n\}$. De constante c is een ondergrens van de meest onderbenaderde norm van een kolom. Neem bijvoorbeeld $c = 0.5$, dan is het mogelijk dat één kolom een kans $p_i = 0.5 \frac{\|D_{:,i}\|_2^2}{\|D\|_F^2}$ heeft en dus met 50% onderschat wordt. Indien $c = 1$ weerspiegelt de kansverdeling exact de echte normen van de kolommen. Deze constante heeft verder invloed op de benaderingsfout.

Deze kansverdeling is van cruciaal belang om theoretisch een garantie te kunnen bieden over de nauwkeurigheid van de benadering die opgesteld is met een willekeurige keuze van kolommen. Het belang hiervan wordt geïllustreerd in volgend

voorbeeld. Stel je voor dat je een lage-rang benadering wil maken van een grote matrix $M \in \mathbb{R}^{n \times n}$ met alle elementen nul buiten één zeer groot element. Om een goede benadering te vinden moet dit element deel uit maken van de gekozen kolommen. Zonder enig idee van waar dit grote element gelegen is in de matrix, is het onmogelijk zeker te zijn dat een willekeurige selectie kolommen dit element bevat. De kansverdeling lost dit probleem op, omdat deze helpt in te schatten in welke kolom de grote elementen schuilen. Voor een afstandsmatrix is het mogelijk om een kansverdeling die voldoet aan (6.23) op te stellen met slechts één rij, dankzij de driehoeksongelijkheid. Dit wordt uitgelegd in Sectie 6.3.2.

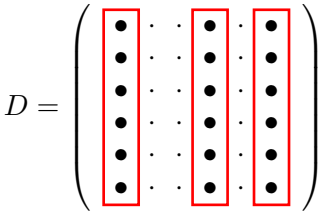
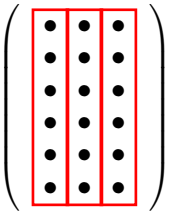
Met een kansverdeling die voldoet aan (6.23) kunnen $s(\ll n)$ kolommen onafhankelijk uit D gekozen worden om zo een matrix S op te stellen. Van de kolomruimte van S kunnen dan de linkse singuliere vectoren $u_1, ..., u_k$ van de k grootste singuliere waarden overgehouden worden. Een projectie van D naar deze vectorruimte (Sectie 2.3.2) voldoet dan theoretisch met waarschijnlijkheid van 90% aan

$$\|D - (\sum_{i}^{k} u_i u_i^T)D\|_F^2 \leq \|D - D_k\|_F^2 + \frac{10k}{cs}\|D\|_F^2 \tag{6.24}$$

of wanneer de vectoren $u_1, ..., u_k$ gegroepeerd worden in een matrix U

$$\|D - UU^T D\|_F^2 \leq \|D - D_k\|_F^2 + \frac{10k}{cs}\|D\|_F^2, \tag{6.25}$$

bewezen in [50]. Dit deel van het gehele SOLRADM algoritme is te vinden in Algoritme 7 en vormt de tweede stap van SOLRADM. Algoritme 7 selecteert $s = \mathcal{O}(k/\varepsilon)$ kolommen van D en vormt de matrix S (weliswaar mits herschaling van de kolommen volgens de kans van de kolom in de kansverdeling) als in

$D =$  $\longrightarrow S =$  .

De vectoren $u_1, ..., u_k$ kunnen berekend worden via de SVD van S met tijdscomplexiteit $\mathcal{O}(n \cdot k^2/\varepsilon^2)$. Als k te groot wordt of ε te klein is, kan het berekenen van de SVD mogelijk te traag zijn. Dit kan aangepakt worden door de Monte-Carlo methode opnieuw uit te voeren, maar op de rijen van S. Dit levert een $s \times s$ matrix op, waarvan de SVD slechts $\mathcal{O}(k^3/\varepsilon^3)$ tijd kost. Deze extra stap gaat theoretisch in ruil voor wat nauwkeurigheid, maar in onze experimenten waren de verschillen verwaarloosbaar klein. Voor het snellere algoritme verwijzen we naar Algoritme 10 in Appendix A.

Na Algoritme 7 beschikken we over een matrix U waarvoor (6.25) geldt. Het product $UU^T D$ kan echter niet uitgerekend worden, aangezien daarvoor nog steeds de volledige matrix D gekend moet zijn. Daarom bestaat de laatste stap uit het

Algorithm 7: Fast Monte-Carlo Low Rank Approximation. Gebaseerd op het beschreven algoritme in [50].

Input: $D \in \mathbb{R}^{n \times n}$, $P = (p_1, p_2, ..., p_n)$, rang k, ε

$s = \frac{10k}{\varepsilon}$

$S = 0^{s \times k}$

for *i = 1,2,..., s* **do**

  Selecteer één x uit $1..n$ volgens P,

  $S_{:,i} = \frac{1}{\sqrt{sp_x}} D_{:,x}$

end

Bereken de SVD van $S = U\Sigma V^T$,

$[u_1, ..., u_k]$ = Bereken top k linkse singuliere vectoren van S,

Result: $U = [u_1, ..., u_k] \in \mathbb{R}^{n \times k}$

benaderen van $U^T D$. Omdat U gekend is, kan dit probleem gereduceerd worden tot een regressieprobleem, verder besproken in Sectie 6.3.3. Eerst kijken we naar hoe de kansverdeling die voldoet aan (6.23) opgesteld kan worden voor een afstandsmatrix D.

6.3.2 Benadering kolomnormen (Stap 1)

Via de driehoeksongelijkheid in een metrische afstandsmatrix D voor tijdreeksen $\mathcal{T} = \{T_1, ..., T_n\}$ kunnen de normen van de kolommen benaderd worden om te voldoen aan (6.23)[49]. Het kwadraat van de 2-norm van een rij $D_{i^*,:}$ (of een kolom $D_{:,i^*}$)

$$\|D_{i^*,:}\|_2^2 = \|D_{:,i^*}\|_2^2 = \sum_{j=1}^{n} d(T_{i^*}, T_j)^2 \tag{6.26}$$

is de som van de kwadratische afstanden van de tijdreeks T_{i^*} tot alle andere tijdreeksen. Het rechterlid van (6.26) delen door n geeft de gemiddelde kwadratische afstand van de tijdreeks T_{i^*} tot de andere tijdreeksen. Door één rij als referentie te nemen, kan de norm van de andere kolommen ingeschat worden als het kwadraat van de afstand tot de referentie, plus de gemiddelde kwadratische afstand van de referentie tot de andere tijdreeksen, nl.

$$\|D_{i,:}\|_2^2 \approx d(T_i, T_{i^*})^2 + \frac{1}{n} \sum_{j=1}^{n} d(T_{i^*}, T_j)^2. \tag{6.27}$$

De kansverdeling op basis van deze geschatte normen voldoet aan (6.23) (bewezen in [49]). Eén rij van de matrix, de i^*-de rij, volstaat dus om via 6.27 de norm van alle andere kolommen goed te benaderen. Algoritme 8 toont het algoritme hiervoor. Onze experimenten in Sectie 7.2.2 tonen aan dat de norm van de referentierij best zo klein mogelijk is. Daarom is het nuttig om Algoritme 8 een aantal keer te herhalen, en de kansverdeling te kiezen van de referentierij met de laagste norm. Onze standaard configuratie van SOLRADM in de softwarebibliotheek [10] voert dit 10 keer uit. Deze aanpassing kost 9 extra bemonsterde rijen.

Algorithm 8: Opstellen van kansverdeling (benadering van normen van de kolommen), gebaseerd op [49]

Input: Afstandsmatrix $D \in \mathbb{R}^{n \times n}$

Kies $i^* \in [n]$ willekeurig uniform.

$P = (p_1, p_2, ..., p_n)$

for $t = 1, ..., n$ **do**

  $p_t = D^2_{t,i^*} + \frac{1}{n} \sum_{j=1}^{n} D^2_{i^*,j}$

end

$P = \frac{P}{\sum P}$

Result: Kansverdeling P

6.3.3 Regressieprobleem (Stap 3)

Na de tweede stap van SOLRADM (Algoritme 7) beschikken we over een matrix $U \in \mathbb{R}^{n \times k}$, die bij benadering de k linkse singuliere vectoren van de k grootste singuliere waarden van D bevat. Omdat D niet volledig gekend is, kunnen we $UU^T D$ uit (6.25) niet uitrekenen. Daarom bestaat de laatste en weliswaar de meest complexe stap van SOLRADM uit het oplossen van het regressieprobleem

$$V^* = [v_1^*, ..., v_n^*] = \underset{V}{\operatorname{argmin}} \|D - UV\|_F^2. \tag{6.28}$$

Dit probleem is equivalent aan het oplossen van een regressieprobleem

$$v_i^* = \underset{V_{:,i}}{\operatorname{argmin}} \|D_{:,i} - UV_{:,i}\|_2^2 \tag{6.29}$$

voor elke overeenkomstige kolom van D en UV. Door te herschrijven volgens $v = V_{:,i}$ en $y = D_{:,i}$ is dit per kolom van D duidelijk te herleiden naar het klassieke probleem

$$v^* = \underset{v}{\operatorname{argmin}} \|y - Uv\|_2^2. \tag{6.30}$$

In essentie wanneer U en y gekend zijn, kan (6.30) opgelost worden via de kleinste-kwadratenmethode. Maar in het geval van dit werk is y een kolom van de afstandsmatrix D en dus moet het aantal elementen van y die berekend moeten worden, klein blijven.

Dit probleem stelt zich ook bij klassieke *data mining* problemen waarbij een matrix $U \in \mathbb{R}^{n \times k}$ gegeven is en bestaat uit n data-elementen met elk k *features*. In de vector y wordt elk data-element verbonden met een *label*. Het probleem is dan het vinden van een functie, in dit geval een matrixvermenigvuldiging met de vector

v, die een transformatie doet van de k features naar een *label*, als volgt:

$$\overbrace{\begin{bmatrix} u_{11} & \cdots & u_{k1} \\ u_{12} & \cdots & u_{k2} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ u_{1n} & \cdots & u_{kn} \end{bmatrix}}^{\text{data-elementen met k features}} \overbrace{\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}}^{\text{functie}} \approx \overbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}}^{\text{labels}}$$

Omdat dit probleem overgedetermineerd ($n > k$) is, zal het hoogstens mogelijk zijn om een functie te vinden die *bijna* alle *labels* juist voorspelt. Omdat het berekenen van een *label* duur kan zijn, bijvoorbeeld omdat een persoon handmatig de data moet bekijken om er een *label* op te plakken, zal men bij voorkeur de functie berekenen met een deel van deze data-elementen. De kunst zit hem dan in het kiezen van de belangrijkste data-elementen. Een voorbeeld is *importance sampling*, waarbij de belangrijke monsters een hogere kans krijgen om gekozen te worden. Een dergelijke kans kan dan bijvoorbeeld hoger liggen voor grenspunten. Bij *actief leren* verandert de kansverdeling bovendien constant op basis van de eerder gekozen monsters. Op dit domein wordt nog zeer actief onderzoek gedaan, en SOLRADM steunt dan ook op de resultaten van een recent werk [51] rond actieve regressie. De techniek in dit laatste werk is één van de eerste die actief voor $\mathcal{O}(k/\varepsilon)$ data elementen een label opvraagt en vervolgens met een grote waarschijnlijkheid in staat is om een benadering $\tilde{v}$ te vinden, waarvoor geldt dat

$$\|U\tilde{v} - y\|_2^2 \leq (1+\varepsilon)\|Uv^* - y\|_2^2 \tag{6.31}$$

met v^* de best mogelijk oplossing uit (6.30) (ook bewezen in [51]).

Het algoritme van actief leren uit [51] werd in deze thesis lichtjes aangepast en vervolgens als Algoritme 9 geïmplementeerd in het gehele SOLRADM algoritme. Algoritme 9 selecteert $\mathcal{O}(k/\varepsilon)$ rijen van U (elke rij is een data-element x_i) op een *goed gebalanceerde* wijze. Voor de functie of transformatievector $\tilde{v}$ die we zoeken, moet eerst een orthonormale basis $[v_1, ..., v_k]$ opgesteld worden.[1] Door de rijen van U te transformeren onder de vectoren van de orthonormale basis (bv. $v_1(x_2)$), krijgt het algoritme een idee over welke rijen gelijkaardig blijven onder de transformatie. Bij de keuze van de rijen wordt de voorkeur gegeven aan rijen die enerzijds niet dichtbij eerder gekozen rijen liggen en anderzijds een grote variantie vertonen onder de mogelijke oplossingen. De *while-loop* in Algoritme 9 stopt na $\mathcal{O}(k/\gamma^2) = \mathcal{O}(k/\varepsilon)$

[1] Voor een U berekend via Algoritme 7 volstaat het om voor deze basis de standaardbasis van $\mathbb{R}^k$ ($[1, 0, 0, ..., 0]$, $[0, 1, 0, ..., 0]$, etc.) te nemen, omdat U zelf orthonormale kolommen heeft. Als U opgesteld werd via de snellere versie Algoritme 10, dan moet de orthonormale basis van $\tilde{v}$ anders genomen worden. We verwijzen daarvoor naar Appendix A. Onze softwarebibliotheek biedt beide versies aan, maar de experimenten zijn gebaseerd op de snelste versie, omdat er amper verschil is in nauwkeurigheid.

iteraties[51] en selecteert evenveel rijen uit U. De overeenkomstige rijen hiervan in D zijn de rijen die uitgerekend moet worden. Elk van deze rijen krijgt ook een gewicht w, waarmee vervolgens per kolom van D het kleinste-kwadratenprobleem opgelost kan worden om $V = [\tilde{v}_1, ..., \tilde{v}_n]$ te bepalen. Het resultaat UV voldoet dan aan (6.22).

Algorithm 9: Regressie op basis van *well-balanced randomized sampling*. Hierin is $Tr(\cdot)$ het product van de elementen op de diagonaal van de matrix (het spoor). Gebaseerd op het werk in [51].

Input: $U \in \mathbb{R}^{n\times k} = [x_1, ..., x_n]^T$, Afstandsmatrix $D \in \mathbb{R}^{n\times n}$, ε

1 Neem P uniforme verdeling over $(x_1, x_2, ..., x_n)$.
2 Bepaal een orthonormale basis $[v_1, ..., v_k]$ over $(x_1, x_2, ..., x_n)$.
3 $\gamma = \frac{\sqrt{\varepsilon}}{3}$, $mid = \frac{4d/\gamma}{1/(1-\gamma)-1/(1+\gamma)}$
4 $j = 0, B_0 = 0^{k\times k}$
5 $l_0 = \frac{-2k}{\gamma}, u_0 = \frac{2k}{\gamma}$
6 **while** $u_j - l_j < \frac{8k}{\gamma}$ **do**
7 $\quad \Phi_j = Tr((u_j I_k - B_j)^{-1}) + Tr((B_j - l_j I_k)^{-1})$
8 $\quad P_j(x) \sim v(x)^T (u_j I_k - B_j)^{-1} v(x) + v(x)^T (B_j - l_j I_k)^{-1} v(x)$ met $v(x) = [v_1(x), ..., v_k(x)]$
9 $\quad r_j =$ willekeurige rij x van U volgens kansverdeling P_j
10 $\quad s_j = \frac{\gamma}{P_j(r_j)}$
11 $\quad w_j = \frac{s_j}{mid}$
12 $\quad B_{j+1} = B_j + s_j \cdot v(r_j) v(r_j)^T$
13 $\quad u_{j+1} = u_j + \frac{\gamma}{\Phi(1-\gamma)}, l_{j+1} = l_j + \frac{\gamma}{\Phi(1+\gamma)}$
14 $\quad j = j + 1$
15 **end**
16 $W = [w_1, ..., w_j]$ (De gewichten)
17 $Y =$ Bereken de rijen van D met dezelfde indices als de monsters $(r_1, ..., r_j)$
18 **for** $i = 1, ..., n$ **do**
19 $\quad \tilde{v}_i =$ Kleinstekwadraten(U, $Y_{:,i}$) met gewichten W
20 **end**

Result: $V = [\tilde{v}_1, ..., \tilde{v}_n]$

Het berekenen van een nieuwe kansverdeling (lijn 8 in Algoritme 9) heeft een tijdscomplexiteit van $\mathcal{O}(n \cdot k^2)$ per iteratie, waardoor de totale complexiteit van Algoritme 9 neerkomt op $\mathcal{O}(n \cdot k^3/\varepsilon)$. Deze stap is ook de doorwegende factor op de gehele tijdscomplexiteit van SOLRADM. Zolang k klein blijft, is dit een acceptabele snelheid (in de experimenten vinden we een goede benadering met $k = 50$ voor een 24000 bij 24000 matrix). SOLRADM gebruikt in totaal $\mathcal{O}(1) + \mathcal{O}(k/\varepsilon) + \mathcal{O}(k/\varepsilon)$ rijen van de matrix, wat de totale tijdscomplexiteit van het berekenen van de afstandsmatrix op $\mathcal{O}(n \cdot (k/\varepsilon) \cdot a)$ brengt met a kost van de evaluatie van de afstandsfunctie voor tijdreeksen.

Hoofdstuk 7

Experimenten

In dit hoofdstuk worden de technieken uit het vorige hoofdstuk geanalyseerd en toegepast in een volledige clusterpijplijn. Voor deze experimenten wordt gebruik gemaakt van de de data van het *UCR Time series Classification Archive*[1]. Sectie 7.1 bespreekt verder deze data. In sectie 7.2 worden ACA en SOLRADM toegepast op de data en wordt een grondige vergelijking gemaakt. Ten slotte wordt de impact van de benadering op de hele clusterpijplijn onderzocht in sectie 7.3.

7.1 UCR Time Series Archive

Het UCR (Universiteit van Californië, Riverside) Time Series (Classification) Archive[1] is een archief met op het moment van schrijven 128 verschillende verzamelingen van tijdreeksen. De verzamelingen zijn opgesteld vanuit erg uiteenlopende domeinen, o.a. agricultuur, schriftherkenning, energieverbruik en bewegingsanalyses. Door deze heterogeniteit is er ook een groot verschil in de grootte van de verzamelingen (40 - 24000 tijdreeksen) en de lengte van de tijdreeksen (24 - 2844).

De grote diversiteit van de datasets maakt het UCR Time Series Archive een uitstekend middel om de effectiviteit van de algoritmes te testen. In de komende experimenten worden 38 van deze datasets gebruikt [1]. Dit werk focust zich op de schaalbaarheid van het clusteren. Daarom gebruiken we enkel datasets met minstens 800 tijdreeksen. Daarnaast laten we ook de datasets met tijdreeksen met missende waarden, de datasets die in principe multivariate tijdreeksen vormen (bv. *CricketX*, *CricketY* en *CricketZ*) en eventueel later toegevoegde datasets, vallen. Alle datasets zijn reeds z-genormaliseerd (Sectie 2.1.1).

[1] CBF, ChlorineConcentration, CinCECGTorso, Crop, ECG5000, ECGFiveDays, ElectricDevices, EthanolLevel, FaceAll, FacesUCR, FiftyWords, FordA, FordB, FreezerRegularTrain, FreezerSmallTrain, HandOutlines, InsectWingbeatSound, ItalyPowerDemand, Mallat, MedicalImages, MixedShapesRegularTrain, MixedShapesSmallTrain, MoteStrain, NonInvasiveFetalECGThorax1, NonInvasiveFetalECGThorax2, PhalangesOutlinesCorrect, Phoneme, ShapesAll, SonyAIBORobotSurface2, StarLightCurves, Strawberry, SwedishLeaf, Symbols, TwoLeadECG, TwoPatterns, Wafer, WordSynonyms, Yoga

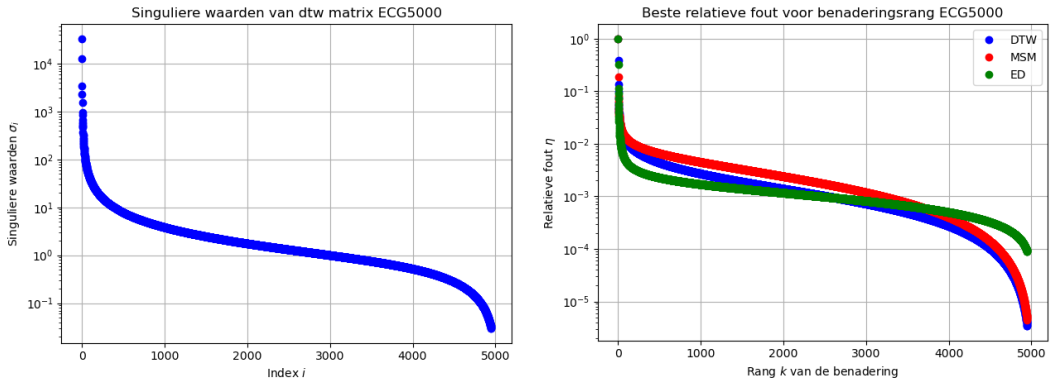
Elk van deze datasets bestaat uit verschillende klassen. Elke tijdreeks is geassocieerd met het nummer van de klasse waartoe zij behoort. Deze klassen vormen een partitie $\mathcal{G} = \{G_1, ..., G_s\}$ van s clusters. De clusters, berekend door een clusteralgoritme, kunnen dan vergeleken worden met $\mathcal{G}$ om te kijken hoe goed het clusteralgoritme presteert.

7.2 Lage-rang benaderingen

De eerste stap van het clusteren van een verzameling van n tijdreeksen bestaat uit het opstellen van de afstandsmatrix $D \in \mathbb{R}^{n \times n}$. Deze matrix kan exact berekend worden of benaderd worden via ACA of SOLRADM. In deze sectie onderzoeken we de kwaliteit van deze algoritmes.

7.2.1 Impact van de afstandsfunctie op de beste benadering

In het eerste experiment kijken we naar de impact van de afstandsfuncties op de afstandsmatrix. Hier zijn we vooral geïntereseerd in de rang k nodig is om een afstandsmatrix goed te benaderen, en dus in de mathematisch best mogelijke benadering. Deze beste benaderingsfout is afhankelijk van de singuliere waarden van de matrix en kan berekend worden via (2.19). De singuliere waarden zullen afhankelijk zijn van de gekozen afstandsfunctie, dus we willen hier ook zien of er een significant verschil is tussen DTW, MSM en ED. In Figuur 7.1 wordt deze vergelijking gemaakt voor de dataset *ECG5000*. Op het eerste gezicht zijn er geen grote verschillen tussen de afstandsfuncties en hebben ze een gelijkaardig verloop van de beste benaderingsfout.



(a) **(b)**

Figuur 7.1. Voor de dataset *ECG5000* (a) de singuliere waarden van de DTW afstandsmatrix en (b) de best mogelijke fout voor DTW-MSM-ED afstandsmatrix in functie van de rang van de benadering.

Het experiment werd vervolgens uitgevoerd op alle 38 datasets, waarbij de beste benaderingsfout werd berekend voor de rangen $k = 5, 10, 20, 50, 100$. Hiervan werd het gemiddelde μ, de standaardafwijking σ en het maximum genoteerd in Tabel 7.1. Uit deze tabel blijkt dat gemiddeld voor alle afstandsfuncties een zeer goede benadering met een relatieve fout kleiner dan 0.02 kan gevonden worden vanaf $k = 50$. Gemiddeld is er geen groot verschil tussen de verschillende afstandsfuncties. Voor de metrische afstandsfuncties MSM en ED liggen de standaardafwijkingen en maxima een beetje lager dan voor DTW. Hoewel de verschillen klein zijn, kunnen we dus verwachten dat er bij DTW een iets grotere onzekerheid zal zijn over de nauwkeurigheid van de gevonden benaderingen.

	DTW			MSM			ED		
Rang k	μ	σ	max	μ	σ	max	μ	σ	max
5	0.066	0.023	0.122	0.056	0.013	0.091	0.062	0.012	0.092
10	0.040	0.017	0.087	0.037	0.008	0.055	0.033	0.007	0.049
20	0.027	0.012	0.056	0.027	0.007	0.043	0.020	0.006	0.037
50	0.017	0.008	0.031	0.018	0.006	0.031	0.011	0.004	0.021
100	0.012	0.006	0.025	0.014	0.005	0.022	0.007	0.003	0.016

Tabel 7.1. De beste benaderingsfout η_{best} van de DTW, MSM en ED afstandsmatrix werd berekend via (2.19) voor alle 38 datasets op verschillende lage rangen k. Voor al deze rangen werd het gemiddelde μ, de standaardafwijking σ en het maximum van de beste benaderingsfout berekend.

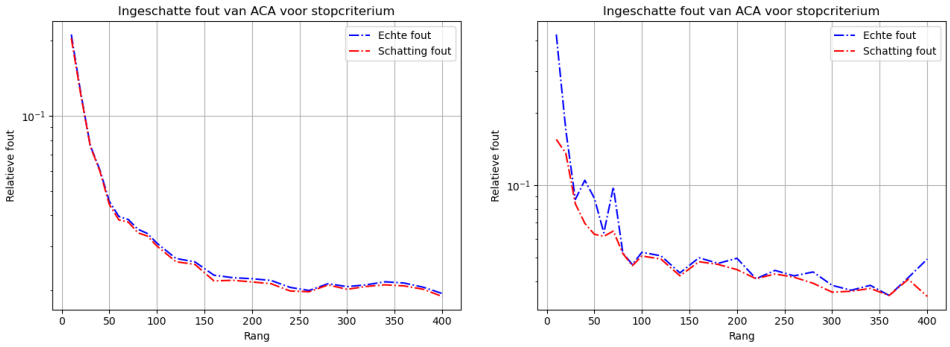
7.2.2 Aanpassingen van de algoritmes ACA en SOLRADM

In deze sectie willen we met een paar experimenten de aanpassingen, die in dit werk voorgesteld werden voor ACA en SOLRADM, verantwoorden. Eerst kijken we bij ACA naar de efficiëntie van de stochastisch benadering van de relatieve fout. Vervolgens voeren we nog een aantal experimenten uit om te zien hoe goed de eerste stap van SOLRADM de kolomnormen inschat op basis van een referentierij. Bij deze experimenten maken we ook een onderscheid tussen de drie afstandsfuncties ED, MSM en DTW.

Stochastische bepaling van de relatieve fout bij ACA

Onze implementatie van ACA, in Sectie 6.2, schat stochastisch de relatieve fout van de benadering in tijdens het uitvoeren van het algoritme. In dit experiment voeren we ACA ($\tau = 0$, $\Delta_k = \infty$) uit op de MSM en de DTW afstandsmatrices van *ECG5000* (5000 tijdreeksen) en bekijken we deze ingeschatte relatieve fout tegenover

de echte relatieve fout terwijl de rang k van de benadering stijgt van 10 tot 500[2]. Het resultaat is te zien in Figuur 7.2. Voor deze stochastische benadering werd slechts een fractie van de matrixelementen gekozen (door Algoritme 4) om de relatieve fout in te schatten, nl. 25000 (5 per rij) voor MSM en 100000 (20 per rij) voor DTW. Dat is beide minder dan 1% van de matrix. Het aantal monsters werd gekozen op basis van (6.20), wat aantoont dat de elementen van DTW een grotere spreiding hadden en dus meer monsters nodig geacht werden voor de stochastische benadering. Wat meteen opvalt in Figuur 7.2 is het verschil tussen MSM en DTW. De stochastische inschatting van de relatieve fout is zeer accuraat voor MSM, maar voor DTW zijn er duidelijk momenten waarbij de relatieve fout serieus fout ingeschat wordt. De resultaten tonen ook aan dat de echte fout meestal onderschat wordt, wellicht door de bias van het algoritme om ook de monsters te gebruiken om de volgende rij van het skelet te selecteren (Sectie 6.2). Daarnaast zien we dat het verloop van ACA met DTW een stuk instabieler is dan met MSM. We verwachten dat ACA slechter zal presteren voor DTW matrices bij verdere experimenten met het volledige algoritme.



(a) MSM **(b)** DTW

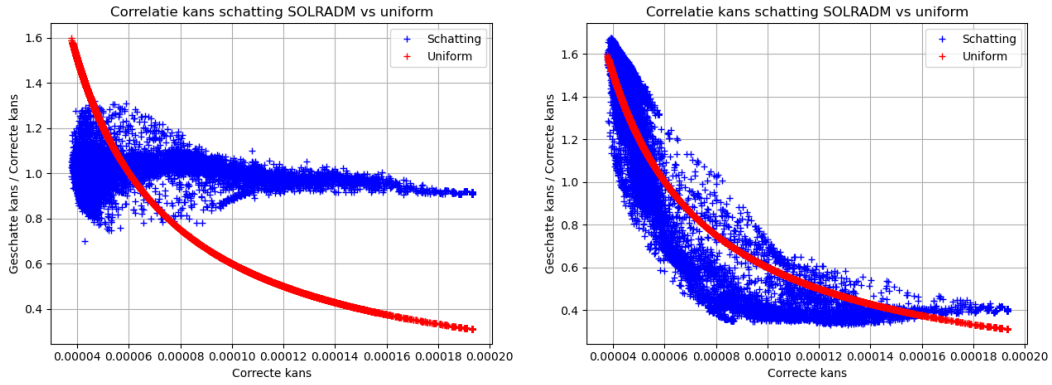
Figuur 7.2. De echte relatieve fout tegenover de stochastisch ingeschatte fout tijdens de uitvoering van ACA op de MSM (a) en DTW (b) afstandsmatrices van *ECG5000*. Voor deze inschatting waren minder dan 1% van de matrixelementen nodig.

Inschatting van de kolomnormen in SOLRADM

De eerste stap van SOLRADM, Algoritme 8, berekent een kansverdeling over de kolommen van de afstandsmatrix D op basis van een benadering van de kolomnormen. Deze verdeling is belangrijk om kolommen met een grote norm (en dus grote elementen) op te merken.

In een eerste experiment worden verschillende kansverdelingen voor de kolommen van de MSM afstandsmatrix van *ElectricDevices* opgesteld via Algoritme 8. Deze

[2] in stappen van 10 onder 100 en stappen van 20 erboven



(a) Goede benadering ($c \approx 0.7$). **(b)** Slechte benadering ($c \approx 0.3$).

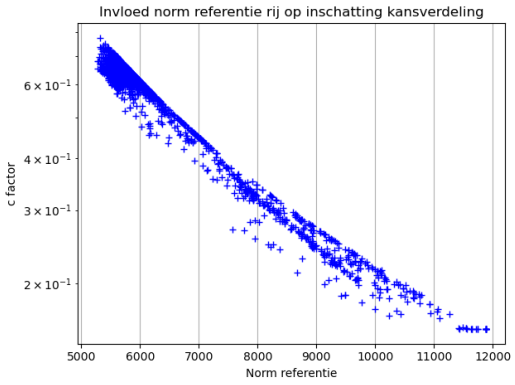
Figuur 7.3. Twee kansverdelingen werden via Algoritme 8 opgesteld voor de MSM afstandsmatrix (16637 × 16637) van *ElectricDevices*. De geschatte kans van elke kolom wordt vergeleken met de correcte kans(blauw). De kansverdeling wordt vergeleken met de uniforme verdeling (rood).

verdeling wordt dan vergeleken met de correcte kansverdeling met de kans van elke kolom

$$p_i = \frac{\|D_{:,i}\|_2^2}{\|D\|_F^2}. \tag{7.1}$$

Voor elke test wordt naar de factor c uit (6.23) gekeken, de maat voor de meest onderschatte kans van een kolom. Deze factor is van theoretisch belang in (6.25). Bij een lage c moet meer overbemonsterd worden om gemiddeld een goede benadering te vinden. De kansverdelingen worden ook vergeleken met de uniforme kansverdeling. Kolommen met kleine en grote normen hebben in de uniforme verdeling een gelijke kans, waardoor er respectievelijk een over- en onderschatting van de normen is (en hierdoor een lage c). Door de willekeurige keuze van de referentierij in Algoritme 8 liggen de resultaten soms ver uit elkaar. Figuur 7.3 toont een voorbeeld van een goede en slechte kansverdeling. De waarde van c kan afgelezen worden als de y-waarde van het laagste meetpunt.

De oorzaak van de slechte kansverdeling zoeken we in de norm van de referentierij in Algoritme 8. Wanneer de norm van de rij met index i^* hoog is ten opzichte van de andere rijen, is de gemiddelde kwadratische afstand van de referentie tot de andere tijdreeksen groot, waardoor de kansverdeling dichter bij de uniforme kansverdeling aanleunt. We onderzoeken dit vermoeden door het voorafgaande experiment 2000 keer uit te voeren en telkens de waarde van c te vergelijken met de norm van de referentierij. Figuur 7.4 toont de resultaten. Deze suggereren dat het beter is om de kansverdeling op te stellen op basis van een referentierij met een zo laag mogelijke norm. Dit bevestigt ons voorstel om Algoritme 8 meermaals op te roepen in Algoritme 6 en de kansverdeling op te stellen op basis van de rij met de laagste norm.



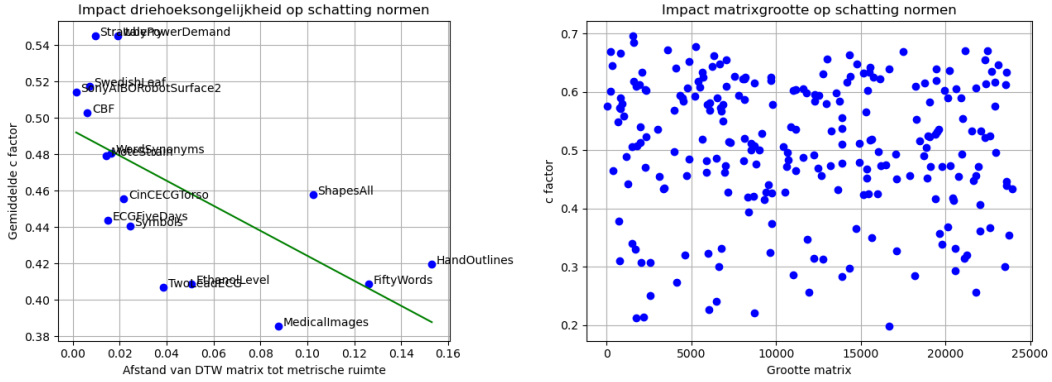
Figuur 7.4. De c factor van 2000 kansverdelingen voor de MSM afstandsmatrix van dataset *ElectricDevices* worden vergeleken met de norm van de referentierij.

We herhalen dit experiment vervolgens 1000 keer voor alle 38 datasets voor zowel ED, MSM als DTW met enerzijds 1 rij en anderzijds 10 rijen als referenties. Bij 10 rijen nemen we de kansverdeling van de referentierij met de kleinste norm. Tabel 7.2 toont het gemiddelde en de standaardafwijking van de c factor. Deze resultaten bevestigen nogmaals het nut van het herhaaldelijk uitvoeren van Algoritme 8 en tonen ook het verschil tussen de afstandsfuncties ED, MSM en DTW. Zoals verwacht zijn de kansverdelingen van DTW matrices slechter dan die van ED en MSM omdat het algoritme steunt op de driehoeksongelijkheid. Voor 10 rijen zijn deze verschillen echter minimaal en is het dus waarschijnlijk dat SOLRADM ook goed werkt voor DTW afstandsmatrices.

Gemiddelde en standaardafwijking c factor						
	ED		MSM		DTW	
	μ	σ	μ	σ	μ	σ
1 rij	0.511	0.124	0.523	0.104	0.479	0.148
10 rijen	0.640	0.108	0.638	0.063	0.631	0.107

Tabel 7.2. Het gemiddelde μ en de standaardafwijking σ van de c factor voor 1000 kansverdelingen opgesteld door Algoritme 8 op 1 rij en 10 rijen op te roepen voor de DTW, MSM en ED afstandsmatrix van alle 38 datasets.

Dit onderzoeken we verder door, voor elke dataset met minder dan 1500 tijdreeksen, voor de DTW afstandsmatrix ook de afstand tot de metrische ruimte te berekenen via Algoritme 3 en formule (5.10). In Figuur 7.5a wordt de gemiddelde c factor (met 1 rij) vergeleken met deze afstand en merken we zoals verwacht op dat voor een 'slecht metrische' DTW matrix de normen dus gemiddeld slechter ingeschat worden.



(a) **(b)**

Figuur 7.5. De c factoren van verschillende kansverdelingen op DTW matrices worden vergeleken met (a) de afstand tot een metrische matrix (berekend via Algoritme 3) en (b) de dimensie van de matrix.

Ten slotte is het voor de schaalbaarheid van SOLRADM belangrijk dat de schatting van de kolomnormen niet beïnvloed wordt door de grootte van de matrix. Daarom gebruikten we de grootste dataset, *Crops* (24000 tijdreeksen), om 250 afstandsmatrices van willekeurig grootte te genereren en voor elk van deze matrices een kansverdeling op te stellen met Algoritme 8. De resultaten zijn te zien in Figuur 7.5b en tonen aan dat de dimensie van de matrix weinig impact heeft.

7.2.3 Vergelijking van de volledige algoritmes

Bij de volgende experimenten bestuderen we de volledige algoritmes van ACA en SOLRADM. Hierbij kijken we allereerst naar de relatieve fout

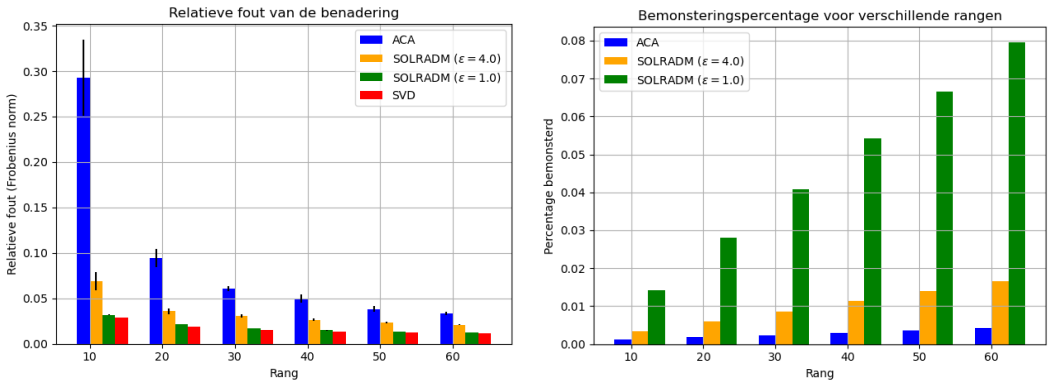
$$\eta = \frac{\|D - \tilde{D}\|_F}{\|D\|_F} \tag{7.2}$$

van de benadering $\tilde{D}$ voor de exacte afstandsmatrix D en vergelijken deze met de best mogelijk fout η_{best} berekend via de SVD en (2.19). Daarnaast is natuurlijk ook het bemonsteringspercentage van belang. Dit bemonsteringspercentage is equivalent aan het aantal keer dat de (dure) afstandsfunctie geëvalueerd moet worden, gedeeld door $\frac{(n+1)n}{2}$ (= aantal elementen in bovendriehoek van de afstandsmatrix).

In het eerste experiment willen we aantonen dat beide algoritmes een goede benadering vinden met een laag bemonsteringspercentage. Voor zowel ACA als SOLRADM stijgt het aantal gekozen rijen (en dus het bemonsteringspercentage) lineair met de rang k van de benadering. Naast de monsters voor het stopcriterium, gebruikt ACA elke bemonsterde rij om een nieuwe rang-één matrix toe te voegen aan de benadering. SOLRADM daarentegen doet aan overbemonstering, die beïnvloed

wordt door de parameter ε (zie Sectie 6.3). Deze parameter is in theorie een bovengrens van de grootste fout, maar onze experimenten tonen aan dat de benaderingen berekend door SOLRADM telkens veel beter zijn dan deze theoretische bovengrens. Daarom gebruiken we ε eerder om de maat van overbemonstering te variëren. In het experiment vergelijken we ACA (met $\tau = 0$), SOLRADM met $\varepsilon = 1.0$ en SOLRADM met $\varepsilon = 4.0$. Voor de MSM afstandsmatrix van *ElectricDevices* werd voor elke rang k (k_{max} in ACA), variërend van 10 tot 60 met stappen van 10, elk algoritme 5 keer uitgevoerd.

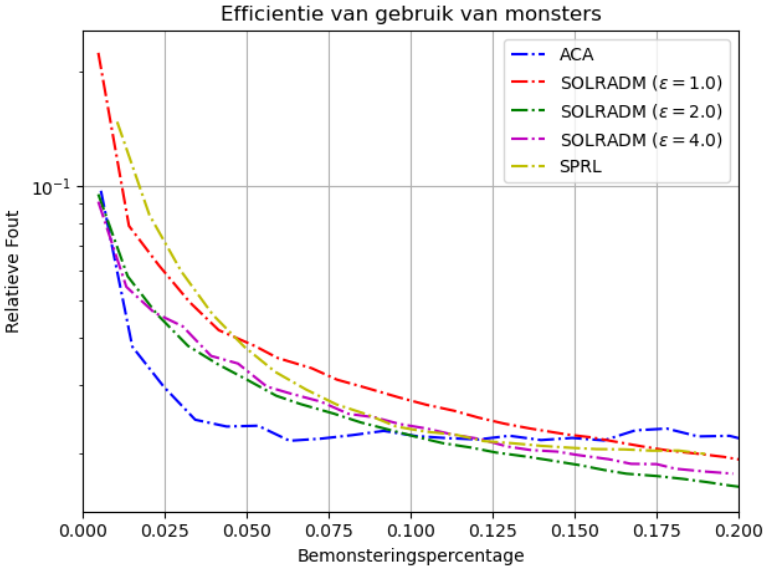
De resultaten van dit experiment staan in Figuur 7.6. Figuur 7.6a toont aan dat de benaderingen opgesteld via SOLRADM met $\varepsilon = 1.0$ zeer dicht bij het optimum liggen, een stuk dichter dan ACA. SOLRADM is duidelijk in staat om betere benaderingen te vinden, maar hierbij moet echter een kanttekening gemaakt worden. Doordat SOLRADM overbemonstert, ligt het bemonsteringspercentage veel hoger dan bij ACA, zoals we zien in Figuur 7.6b. We zien tevens dat voor zowel ACA als SOLRADM het bemonsteringspercentage lineair stijgt met de rang. De kleine standaardafwijkingen in Figuur 7.6a tonen ook aan dat voornamelijk SOLRADM zeer stabiel is. Door het grote verschil in bemonsteringspercentage is het niet mogelijk om te zeggen dat SOLRADM beter werkt dan ACA.



(a) **(b)**

Figuur 7.6. Voor de MSM afstandsmatrix van *ElectricDevices* werden benaderingen bepaald via ACA en SOLRADM. Deze worden vergeleken op basis van (a) de relatieve fout en (b) het bemonsteringspercentage. De zwarte strepen in Figuur 7.6a geven de standaardafwijking aan.

We willen ook graag weten welk algoritme nu het efficiëntst werkt en de gekozen monsters optimaal gebruikt. Daarom herhalen we het vorige experiment op de MSM afstandsmatrix, maar we beginnen met een rang k van 2 en verhogen die met kleine stappen tot het resulteert in een bemonsteringspercentage van 20%. Deze methodiek levert een veel nauwkeurigere kijk op de relatieve fout tegenover het bemonsteringspercentage. In dit experiment vergelijken we ACA en SOLRADM tevens met het



Figuur 7.7. De efficiëntie van ACA, drie versies van SOLRADM en SPRL bij het benaderen van de MSM afstandsmatrix van dataset *ECG5000*. Hoe lager het verloop van de grafieken, des te minder monsters van de matrix nodig zijn om een goede benadering te vormen.

algoritme SPRL. Voor SPRL berekenen we de gelijkenismatrix zoals beschreven in Sectie 3.1 (variërend aantal monsters) en zetten deze terug om naar een afstandsmatrix. Omdat er behoorlijk wat benaderingen berekend moesten worden, werd dit experiment uitgevoerd op de iets kleinere dataset *ECG5000* van 5000 tijdreeksen.

Figuur 7.7 toont de resultaten van dit experiment. ACA heeft de hoogste efficiëntie wanneer het bemonsteringpercentage heel laag ligt. Dit valt te verklaren door de meer adaptieve keuze van rijen van ACA. SOLRADM is minder adaptief en steunt op het bemonsteren van meerdere rijen via de kansverdeling en zal daardoor initieel meer rijen nodig hebben om goed te benaderen. Als het bemonsteringspercentage (en de rang) stijgt, bereikt ACA een plateau en zal de relatieve fout niet verder zakken. SOLRADM daarentegen blijft bij een hoger bemonsteringspercentage de relatieve fout verbeteren en blijft in de buurt van de best mogelijke SVD benadering (Figuur 7.6a). Uit dit experiment blijkt ook dat $\varepsilon = 2.0$ een goede keuze lijkt voor SOLRADM. SOLRADM met $\varepsilon = 2.0$ is over de ganse lijn efficiënter dan SPRL. Als we ACA vergelijken met SPRL zien we dat deze laatste, vanaf een bepaald bemonsteringspercentage, efficiënter wordt doordat ACA een plateau bereikt. Maar de verschillen in relatieve fout blijven beperkt. Deze resultaten suggereren dat ACA zeer efficiënt werkt voor benaderingen met een heel lage rang. Indien meer nauwkeurigheid nodig is, kan er overgestapt worden op SOLRADM.

In een laatste experiment testen we ACA en SOLRADM met vaste configuraties

voor de MSM en DTW afstandsmatrices van alle 38 datasets. Voor ACA testen we met $\tau = 0.05$ en $\tau = 0.02$ (en $k_{max} = \infty$). Dit wil zeggen dat ACA stopt van zodra het inschat dat de benadering een relatieve fout van respectievelijk 0.05 of 0.02 heeft. We laten ACA ook stoppen van zodra er na 100 iteraties geen betere benadering gevonden wordt ($\Delta_k = 100$). Voor elke benadering wordt de relatieve fout en het bemonsteringspercentage bijgehouden. Het bemonsteringspercentage geeft een notie van tijd, omdat het aangeeft hoe vaak de afstandsfunctie geëvalueerd wordt. Bij een bemonsteringspercentage van 1% is het benaderen van de matrix dus 100 keer sneller dan het berekenen van de volledige matrix. Voor een kleine matrix zal het bemonsteringspercentage doorgaans hoger liggen dan voor een grote matrix. Daarom leiden we uit het bemonsteringspercentage ook de bemonsteringsfactor af. Deze factor definiëren we als het aantal bemonsterde elementen van de matrix gedeeld door de grootte n van de dataset. Deze factor geeft een beter idee over de schaalbaarheid van het algoritme.

De volledige resultaten van ACA zijn te vinden in Appendix B in Tabel B.1 en Tabel B.2, en zijn samengevat in Tabel 7.3. Hoewel ACA geen gebruik maakt van de driehoeksongelijkheid merken we toch een groot verschil in de kwaliteit bij de benaderingen van MSM ten opzichte van DTW matrices. Zowel de gemiddeldes als standaardafwijkingen liggen een stuk hoger bij DTW. We zien dat een aantal uitschieters hiervan de oorzaak zijn. Zo is de relatieve fout van 0.2833 bij *HandOutlines* een stuk hoger dan de gewenste fout van 0.05. Deze uitschieters zijn wellicht het gevolg van een foute inschatting van de relatieve fout zoals we ook zagen in Figuur 7.2b. Ook de bemonsteringsfactor van 416.4 ligt bij *HandOutlines* ver boven het gemiddelde. Voor MSM zijn er veel minder van deze uitschieters, wat opnieuw aantoont dat er meer onzekerheid is over de nauwkeurigheid van de DTW benadering.

Voor de meeste datasets heeft ACA wel goede resultaten. Voor de grootste dataset *Crop* is slechts 0.19% en 0.35% van respectievelijk de MSM en DTW afstandsmatrix nodig om een lage-rang benadering met relatieve fout $\eta \approx 0.05$ op te stellen. Door de uitschieters met slechte benaderingen is het toch iets moeilijker om volledig vertrouwen te hebben in ACA.

	ACA ($\tau = 0.05$)				ACA ($\tau = 0.02$)			
	MSM		DTW		MSM		DTW	
	η	$\times n$	η	$\times n$	η	$\times n$	η	$\times n$
μ	0.0523	40.7	0.0615	125.6	0.0235	110.7	0.0358	220.0
σ	0.0015	17.3	0.0142	66.6	0.0033	34.5	0.0182	92.5

Tabel 7.3. Het gemiddelde en de standaardafwijking van de relatieve fout η en de bemonsteringsfactor $\times n$ over de benadering gegenereerd door twee versies van ACA op alle 38 datasets.

Hetzelfde experiment werd ook uitgevoerd met SOLRADM (met $\varepsilon = 2.0$) voor de DTW en MSM matrices van alle 38 datasets. De volledige resultaten staan in Appendix B in Tabel B.3 en Tabel B.4 en de samenvatting staat in Tabel 7.4. In tegenstelling tot ACA, is de gemiddelde relatieve fout van een MSM en DTW benadering wel vergelijkbaar bij SOLRADM. Dit is verbazend omdat SOLRADM theoretisch enkel geschikt is voor de metrische afstandsfunctie MSM. Aan de standaardafwijking zien we wel opnieuw dat de benaderingen voor DTW meer gespreid liggen.

De bemonsteringsfactor is voor SOLRADM heel stabiel over alle datasets voor zowel $k = 20$ als $k = 50$, waardoor de *runtime* zeer goed voorspelbaar is (want het aantal evaluaties van een afstandsfunctie is n keer de bemonsteringsfactor). Dit is ook wat we verwachten omdat voor een bepaalde rang k en $\varepsilon = 2.0$ SOLRADM $\mathcal{O}(k/\varepsilon)$ rijen zal evalueren. We verwachten dat, als de rang k opgevoerd wordt van 20 naar 50, het aantal bemonsterde rijen en dus de bemonsteringsfactor toeneemt met een factor 2.5. Dit is het geval als we in Tabel B.3 kijken naar grote datasets zoals *Crop* (van 114.2 naar 285.8), *ElectricDevices* (van 114.6 naar 284.1) en *StarLightCurves* (van 112.8 naar 280.2). Voor een kleine dataset zoals WordSynonims gaat de bemonsteringsfactor echter slechts van 103.5 naar 224.4 ($\approx \times 2.2$). Dit valt te verklaren doordat er in de 3 stappen van SOLRADM dezelfde rij meerdere keren gekozen kan worden (en maar één keer moet geëvalueerd worden) en deze kans is veel groter voor een kleine dataset.

Uit dezelfde tabellen blijkt ook duidelijk dat er weinig correlatie is tussen de grootte van de dataset en de relatieve fout. Deze relatieve fout is afhankelijk van de onderliggende data. Dat kunnen we bijvoorbeeld zien aan datasets zoals *FordA* (4921 tijdreeksen) en *FordB* (4446 tijdreeksen). De tijdreeksen in deze datasets zijn verschillend, maar zijn in dezelfde context opgesteld. De benaderingen voor beide datasets hebben zeer gelijkaardige relatieve fouten, nl. 0.0223 vs. 0.0229 (MSM) en 0.0341 vs. 0.035 (DTW) voor $k = 50$.

Uit deze experimenten blijkt dat SOLRADM perfect schaalbaar is voor grote matrices en een stabiele output heeft. We beschouwen SOLRADM dus als een beter algoritme dan ACA en zullen integratie van SOLRADM in de volledige clusterpijplijn verder bestuderen in Sectie 7.3.

	SOLRADM ($k = 20$)				SOLRADM ($k = 50$)			
	MSM		DTW		MSM		DTW	
	η	$\times n$	η	$\times n$	η	$\times n$	η	$\times n$
μ	0.0310	108.8	0.0328	109.0	0.0201	252.0	0.0200	252.8
σ	0.0062	3.8	0.0114	3.4	0.0050	19.1	0.0083	18.2

Tabel 7.4. Het gemiddelde en de standaardafwijking van de relatieve fout η en de bemonsteringsfactor $\times n$ over de benadering gegenereerd door twee versies van SOLRADM op alle 38 datasets.

7.3 Volledige clusterpijplijn

We testen de impact van het gebruik van een benaderde afstandsmatrix in de volledige clusterpijplijn. Allereerst berekenen we voor elke dataset de exacte afstandsmatrix. We doen dit met DTW en MSM als afstandsfuncties. Deze matrix gebruiken we als input voor een clusteralgoritme om een partitie $\mathcal{C} = \{C_1, ...C_s\}$ met s clusters te berekenen. De ARI-score (Sectie 2.2.2) van de exacte matrix ARI_{exact} kan dan berekend worden voor de partitie $\mathcal{C}$ ten opzichte van de echte klassen $\mathcal{G}$. We doen hetzelfde voor de benaderde afstandsmatrix en krijgen dan een score $ARI_{benadering}$. We verwachten dat gemiddeld $ARI_{benadering} < ARI_{exact}$ door de relatieve fout van de benadering, maar we hopen dat als de relatieve fout klein genoeg is, dat gemiddeld $ARI_{benadering} \approx ARI_{exact}$.

7.3.1 Vergelijking over alle datasets

De veelbelovende resultaten uit Sectie 7.2.3 tonen aan dat SOLRADM steeds in de buurt komt van de beste benadering. In het eerste experiment genereren we voor alle 38 datasets via SOLRADM (met $\varepsilon = 2.0$) benaderingen van rang $k = 5, 10, 20$ en 50 voor zowel de DTW als de MSM afstandsmatrix. Deze benaderingen gebruiken we als input voor *complete linkage HAC* en spectraal clusteren (Sectie 2.2.1). Vervolgens wordt voor elke test $ARI_{benadering}$ met ARI_{exact} vergeleken.

De resultaten zijn gevisualiseerd in Figuur 7.8 voor DTW en in Figuur 7.9 voor MSM. Elk punt in de figuren stelt één test van een dataset voor. De rode lijn is waar $ARI_{benadering} = ARI_{exact}$. Punten die linksboven deze lijn liggen, zijn tests waar met de benaderde afstandsmatrix betere clusters gevonden werden dan met de exacte matrix. Dit kan toevallig zijn, maar een andere verklaring zou kunnen zijn dat de lage-rang benadering ruis heeft weggefilterd uit de exacte matrix. We zien bijvoorbeeld dat de dataset *Mallet* bij HAC herhaaldelijk een betere ARI-score behaalt via de benaderde DTW afstandsmatrix. Meestal liggen de meetpunten echter onder de rode lijn en heeft de benaderde matrix dus een lagere ARI-score, zoals verwacht. De groene lijn is de regressielijn door deze meetpunten. Als deze lijn dicht bij de rode lijn ligt, hebben de benaderde matrices dus bijna even goede clusters als de exacte matrices.

We zien in het algemeen dat voor spectraal clusteren deze lijnen bij stijgende rang k al snel overlappen voor DTW en voor MSM. We kunnen dus concluderen dat spectraal clusteren zeer goed werkt in combinatie met een lage-rang benadering van de afstandsmatrix. Ook voor HAC komen de rode en groene lijn relatief dicht bij elkaar in de buurt voor rang 50, maar omdat HAC minder goed in staat is om de juiste clusters te vinden van onze datasets (de meeste ARI_{exact} zijn laag), kunnen we ons minder uitspreken over dit resultaat.

Het gemiddelde van de relatieve fouten voor elke rang uit vorig experiment is opgelijst in Tabel 7.5. Door deze tabel te combineren met de figuren, tonen we dus aan dat de rang-50 benadering met een gemiddelde relatieve fout van 0.02 in staat is om goede clusters af te leveren. Voor de dataset *Crop* wil dit zeggen dat circa. 2.5% van de afstandsmatrix evalueren volstaat om goede clusters te vinden (Tabellen B.3 en B.4). Hierdoor kan de matrix 40 keer sneller berekend worden.

Gemiddelde relatieve fout η		
Benaderingsrang	DTW	MSM
5	0.083	0.068
10	0.050	0.045
20	0.033	0.031
50	0.020	0.020

Tabel 7.5. Gemiddelde relatieve fout van de benaderingen via SOLRADM uit Figuren 7.8 en 7.9.

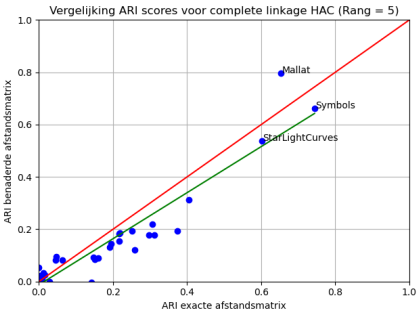
7.3.2 Relatieve fout vs. ARI-score

In het vorige experiment heeft de dataset *Symbols* (1020 tijdreeksen), met zowel HAC als spectraal clusteren, een goede ARI_{exact} score van ongeveer 0.74 met DTW als afstandsfunctie. De tijdreeksen uit verschillende clusters in deze dataset zijn wellicht goed te onderscheiden van elkaar. Daarom onderzoeken we deze dataset nog verder en kijken we meer in detail naar de impact van de relatieve fout van de benadering. In het vorige experiment kon DBSCAN niet geïncludeerd worden, omdat de parameters ε en $minPts$ per dataset en per afstandsfunctie aangepast moeten aangepast worden (Sectie 2.2.1). Dit doen we voor de *Symbols* dataset wel, met als resultaat een ARI_{exact} van 0.78 met $\varepsilon = 2.5$ en $minPts = 10$ voor de DTW afstandsmatrix. In dit experiment gebruiken we ACA en laten we de gewenste relatieve fout τ variëren. Op deze manier hebben we verschillende benaderingen met allemaal verschillende relatieve fouten. Zo genereren we met ACA, met een τ gespreid tussen 0.1 en 0.005, 300 benaderingen van deze DTW afstandsmatrix. Voor deze benadering wordt telkens $ARI_{benadering}$ bepaald en de verhouding $ARI_{benadering}/ARI_{exact}$. Deze verhouding wordt dan vergeleken met de echte relatieve fout η van de benadering.

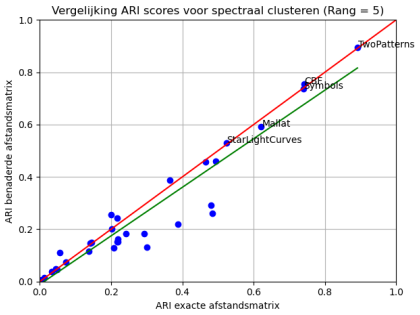
De resultaten zijn weergegeven in Figuur 7.10. Als we kijken naar het bereik van de y-as in Figuur 7.10c, merken we op dat DBSCAN zeer gevoelig is voor de relatieve fout van de benadering. Een relatieve fout van de benadering zal de afstanden zodanig veranderen dat sommige tijdreeksen niet meer voldoen aan de dichtheidsvoorwaarden van DBSCAN (Figuur 2.6). De keuze van ε en $minPts$ gebeurde namelijk op basis van de exacte matrix en is daarom misschien niet goed *getuned* voor de benaderde matrix. We zien wel dat als de relatieve fout klein is (< 0.02) de verhouding naar 1.0 nadert. In de andere figuren wordt opnieuw bevestigd dat spectraal clusteren

goed werkt met een benaderde matrix. Zelfs met de benaderingen met een relatieve fout van 0.05 presteert spectraal clusteren even goed als met de exacte matrix.

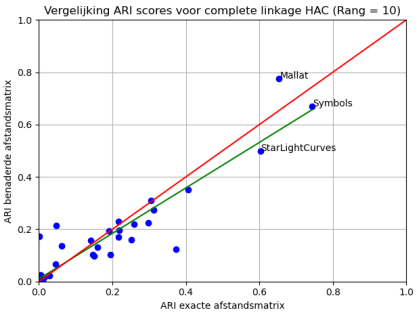
We concluderen hieruit dat het idee van de afstandsmatrix benaderen vooral aantrekkelijk is wanneer het doel is om spectraal te clusteren. DBSCAN en HAC, waar de afstanden tussen de tijdreeksen rechtstreeks impact hebben op de gevormde clusters, zijn gevoeliger voor een relatieve fout van de benadering en eisen dus dat deze fout klein blijft. Door de min of meer instabiele resultaten van ACA, is het moeilijk zeker te zijn dat een ACA benadering een dergelijke kleine fout heeft. De toepassing van ACA lijkt ons daarom vooral in combinatie met spectraal clusteren. Benaderingen via SOLRADM komen dicht in de buurt van de beste benadering en bovendien is de *runtime* erg voorspelbaar. In onze ogen heeft SOLRADM dus het meest potentieel om verder in praktijk gebruikt te worden in combinatie met verschillende clusteralgoritmes.



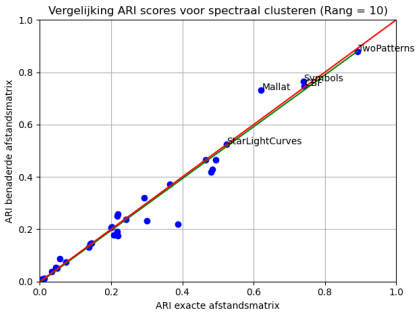
(a) Rang 5, HAC



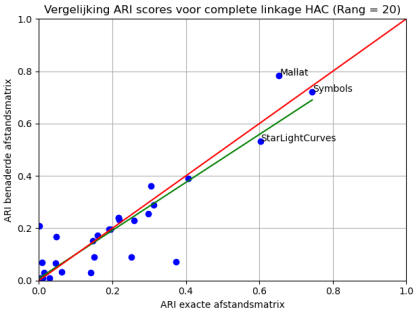
(b) Rang 5, Spectraal



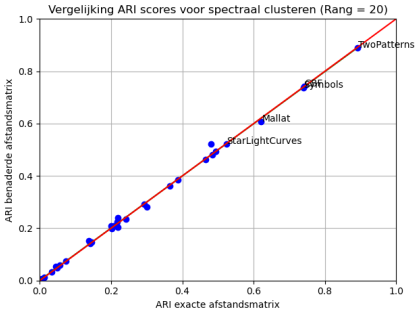
(c) Rang 10, HAC



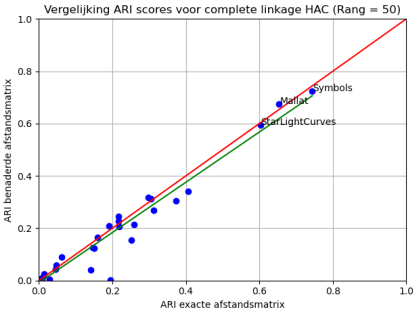
(d) Rang 10, Spectraal



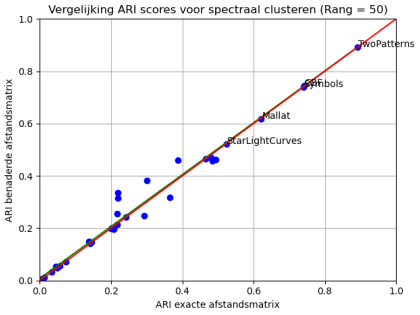
(e) Rang 20, HAC



(f) Rang 20, Spectraal

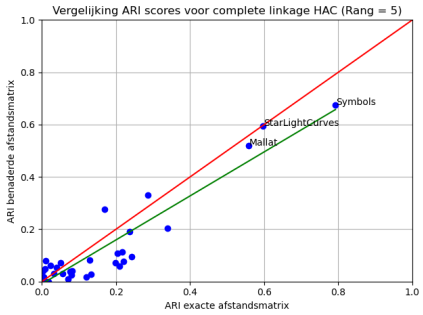


(g) Rang 50, HAC

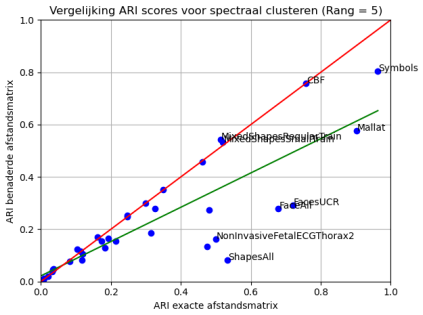


(h) Rang 50, Spectraal

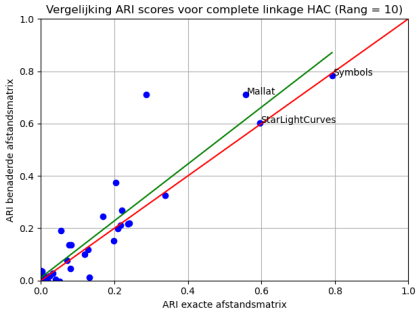
Figuur 7.8. Cluster performantie van SOLRADM benadering van DTW afstandsmatrix. Regressielijn (groen), $ARI_{benadering} = ARI_{exact}$-lijn (rood).



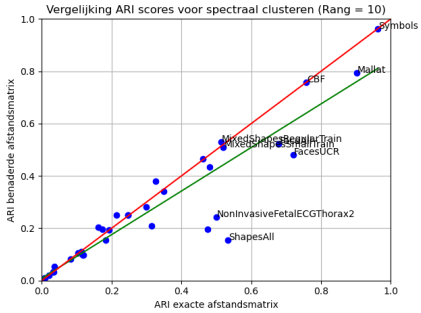
(a) Rang 5, HAC



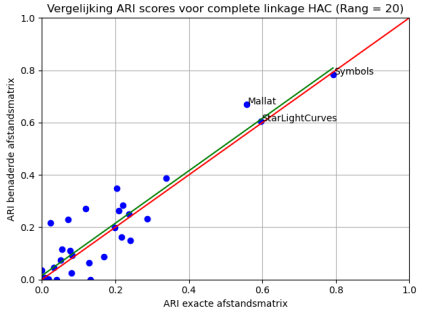
(b) Rang 5, Spectraal



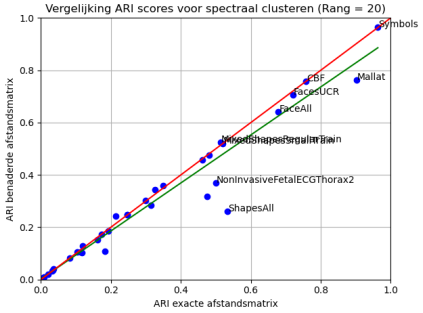
(c) Rang 10, HAC



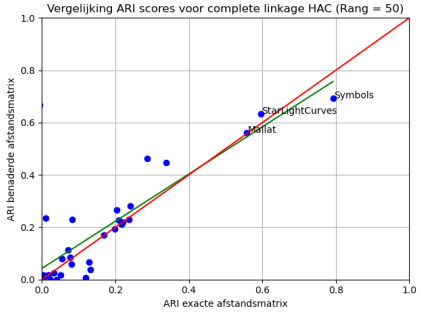
(d) Rang 10, Spectraal



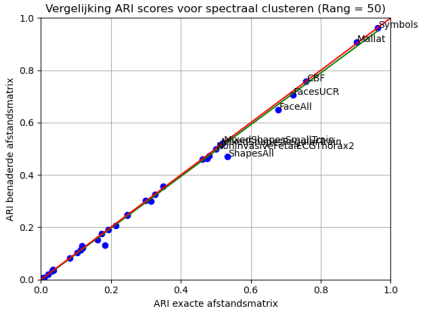
(e) Rang 20, HAC



(f) Rang 20, Spectraal

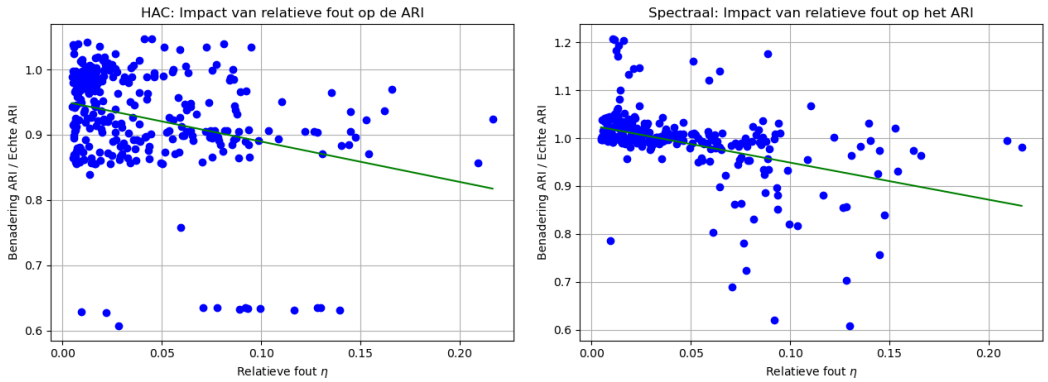


(g) Rang 50, HAC

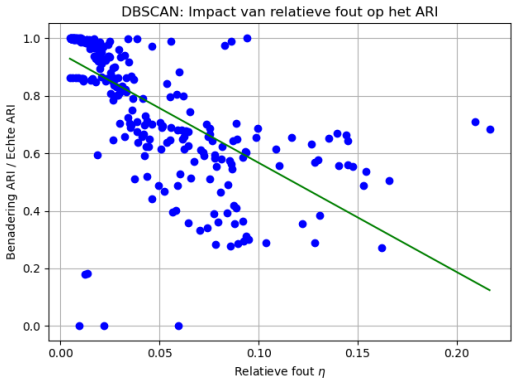


(h) Rang 50, Spectraal

Figuur 7.9. Cluster performantie van SOLRADM benadering van MSM afstandsmatrix. Regressielijn (groen), $ARI_{benadering} = ARI_{exact}$-lijn (rood).



(a) HAC **(b)** Spectraal clusteren



(c) DBSCAN

Figuur 7.10. Correlatie tussen de relatieve fout van de benaderde afstandsmatrix en het verschil in ARI tussen de partitie opgesteld met de echte en de benaderde matrix voor *Symbols*.

Hoofdstuk 8

Besluit

8.1 Conclusies

Het clusteren van n tijdreeksen kan grondig versneld worden door de afstandsmatrix te benaderen via een lage-rang benadering. Het opstellen van de benadering vereist $\mathcal{O}(n/k)$ keer minder (dure) evaluaties van de afstandsfuncties en versnelt bijgevolg significant het gehele clusterproces.

De bijdrage van deze thesis zijn twee algoritmes om een dergelijke lage-rang benadering op te stellen, beschikbaar in de softwarebibliotheek [10]. De eerste methode, ACA, voegt iteratief rang-één matrices toe aan de benadering zodanig dat de benaderingsfout steeds kleiner wordt, tot een opgegeven gewenste maximale benaderingsfout bereikt is. De tweede en in onze ogen de beste methode, SOLRADM, maakt gebruik van de driehoeksongelijkheid om snel de grote kolommen van de afstandsmatrix te situeren en zo te helpen bij het selecteren van kolommen om een goede benadering op te stellen. SOLRADM steunt op een zeer recent onderzoek en onze implementatie is voorzover geweten de eerste publiek beschikbare implementatie van dat onderzoek. De softwarebibliotheek van dit werk biedt ook de implementatie aan van MSM, een metrische afstandsfunctie, gelijkaardig aan DTW. De experimenten tonen aan dat SOLRADM goed werkt voor MSM matrices en, tot onze verbazing, ook voor DTW matrices. SOLRADM en ACA gebruiken $\mathcal{O}(k \cdot n)$ matrixelementen om een rang-k benadering op te stellen, significant minder dan de $\mathcal{O}(n^2)$ van de exacte matrix.

In de experimenten op 38 verschillende datasets in Hoofdstuk 7 tonen we aan dat voor benaderingen met een relatieve fout van 0.02 de gevonden clusters kwalitatief bijna even goed zijn als de clusters gevonden met de exacte matrix. Vooral spectraal clusteren blijkt zeer goed te werken met een lage-rang benadering. ACA en SOLRADM zijn beiden in staat om benaderingen te vinden met een dergelijke kleine relatieve fout, maar SOLRADM is stabieler en zal zelden een slechte benadering geven. Over al onze datasets met MSM en DTW als afstandsfunctie heeft SOLRADM een gemiddelde relatieve fout van 0.02 voor rang-50 benaderingen. Voor het opstellen van deze benadering zijn gemiddeld over alle datasets $252.4 \cdot n$ elementen van de

matrix nodig. Voor de grootste dataset van 24000 tijdreeksen was dit aantal 2.38% van de gehele matrix. De benadering kon dus meer dan 40 keer sneller berekend worden dan de volledige exacte matrix.

8.2 Future work

Om aan te tonen dat de voorgestelde methodes in het algemeen goed werken, werden zowel verschillende afstandsfuncties als verschillende clusteralgoritmes gebruikt voor de experimenten. Dit sterkte ons in onze overtuiging dat voornamelijk SOLRADM een algemeen toepasbaar algoritme is.

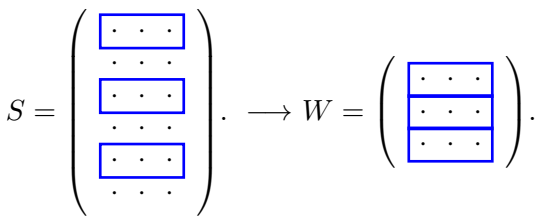
Toekomstig onderzoek zou zich meer kunnen focussen op een specifieke combinatie van afstandsfunctie en clusteralgoritme. Dit werk toonde, een beetje tegen onze verwachting in, dat SOLRADM goed werkt in combinatie met DTW. Met het algoritme om de afstand tot de ruimte van de metrische afstandsmatrices te berekenen, werd aangetoond dat het benaderen van de kolomnormen wel degelijk moeilijker is voor 'slecht' metrische DTW matrices. Door de hoge behoefte aan geheugen van dit algoritme, werd dit enkel getest voor middelgrote matrices. Verder onderzoek zou dit werk kunnen verderzetten op grotere en misschien 'nog slechtere' metrische DTW matrices. Daarnaast zijn er door de populariteit van DTW talloze extra toepassingen met DTW, die mogelijk dus ook in combinatie met SOLRADM getest kunnen worden.

Bijlagen

Bijlage A

Snellere versie van Fast Monte-Carlo Low Rank Approximation

Algoritme 7 kan sneller gemaakt worden door $S \in \mathbb{R}^{n \times s}$ verder te reduceren naar een matrix $W \in \mathbb{R}^{s \times s}$ door rijen te selecteren als (weliswaar mits herschaling)



Het berekenen van de SVD van W heeft een tijdscomplexiteit $\mathcal{O}(s^3)$, lager dan de $\mathcal{O}(n \cdot s^2)$ tijdscomplexiteit voor S.

Het doel van Algoritme 7 is om de k belangrijkste linkse singuliere vectoren ($\mathbb{R}^n$) van D te benaderen via S. Door de verdere reductie naar W hebben de singuliere vectoren ($\mathbb{R}^n$) van W een dimensie s in plaats van n. Het is echter mogelijk om via W de linkse singuliere vectoren van S te benaderen op een analoge manier als S dat doet voor de matrix D. Door rijen te selecteren, zal de ruimte opgespannen door de top rechtse singuliere vectoren van W bij benadering gelijk zijn aan de ruimte opgespannen door de top rechtse singuliere vectoren van S. Uit Sectie 2.3.2 weten we dat $Sv_i = \sigma_i(S)u_i$, dus kunnen we de linkse singuliere vectoren van S benaderen door de rechtse singuliere vectoren van W te transformeren onder S. Via dit principe tonen we de snellere versie van Algoritme 7 in Algoritme 10. Opnieuw moet er eerst een kansverdeling opgesteld worden over de rijen. Hierbij moeten de conditionele kansen van de gekozen kolommen van S ook in acht genomen worden, zoals besproken in [50].

In Algoritme 9 moet een orthonormale basis $[v_1, ..., v_k]$ voor de oplossing $\tilde{v}$ bepaald worden over alle rijen van U. Met de 'over alle rijen van U' wordt bedoeld dat het

Algorithm 10: Snellere versie van Fast Monte-Carlo Low Rank Approximation. Gebaseerd op het beschreven algoritme in [50].

Input: $D \in \mathbb{R}^{n \times n}$, $P = (p_1, p_2, ..., p_n)$, rang k, fout ε

$s = \frac{10k}{\varepsilon}$
$S = 0^{n \times s}$
for *i = 1,2,..., s* **do**
  Selecteer één x uit 1..n volgens P,
  $S_{:,i} = \frac{1}{\sqrt{sp_x}} D_{:,x}$
end
Bereken de SVD van $S = U\Sigma V^T$,
for *i = 1,2,..., s* **do**
  for *j = 1,2,...,n* **do**
    $q_{j|i} = \frac{S_{j,i}^2}{p_i \cdot \|S_{:,i}\|_2^2}$
  end
end
for *j = 1,2,...,n* **do**
  $q_j = \sum_{i=1}^{s} q_{j|i}$
end
Normeer $Q = (q_1, q_2, ..., q_s)$ (kansverdeling over rijen)
$W = 0^{s \times s}$
for *i = 1,2,..., s* **do**
  Selecteer één x uit 1..n volgens Q,
  $W_{i,:} = \frac{1}{\sqrt{sq_x}} S_{x,:}$
end
$[v_1, ..., v_k]$ = Bereken k rechtse singuliere vectoren van W,
for *i = 1,2,..., k* **do**
  $u_i = Sv_i$
  $u_i = u_i / \|u_i\|_2$
end

Result: $U = [u_1, ..., u_k] \in \mathbb{R}^{n \times k}$

inwendig product tussen de basisvectoren voldoet aan

$$\mathop{\mathbb{E}}_{x \sim rijen(U)}[v_i(x) \cdot v_j(x)] = 1_{i=j}[51] \tag{A.1}$$

met $\mathbb{E}[\cdot]$ de stochastische verwachtingswaarde. Een matrix U gegeneerd via Algoritme 7 is altijd orthonormaal, waardoor de vectoren van de eenheidsmatrix altijd voldoen aan A.1. Echter, bij het gebruik van de snellere versie, Algoritme 10, is dit niet meer het geval. Daarom moet de orthonormale basis specifiek berekend worden. Dit kan door de Gram-Smidt orthogonalisatie aan te passen met het inwendig product tussen twee vectoren v_i en v_j gedefinieerd als $\mathop{\mathbb{E}}_{x \sim rijen(U)}[v_i(x) \cdot v_j(x)]$. Dit wordt voorgesteld in Algoritme 11.

Algorithm 11: Variant van Gram-Smidt orthogonalisatie om een orthonormale basis te berekenen over de rijen van een matrix.

Input: $U = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^{n \times k}$
$[v_1, ..., v_k] = [[1, 0, ..., 0]^T, ..., [0, ..., 0, 1]^T]$
for *i = 1,2,..., k* **do**
  for *j = 1, ..., i* **do**
    $e_j = \mathop{\mathbb{E}}_{x \sim rijen(U)}[v_i(x) \cdot v_j(x)]$
  end
  $v_i = v_i - \sum_{m=1}^{i-1} e_m v_m$
  $v_i = \frac{v_i}{\mathop{\mathbb{E}}_{x \sim rijen(U)}[v_i^2(x)]}$
end
Result: Orthonormale basis $[v_1, ..., v_k]$

Bijlage B

Resultaten experimenten

Benadering van MSM afstandsmatrix via ACA							
		ACA ($\tau = 0.05$)			ACA ($\tau = 0.02$)		
Dataset	n	η	%	$\times n$	η	%	$\times n$
CBF	930	0.0504	6.29	29.3	0.0204	14.63	68.1
ChlorineConcentration	4307	0.0542	1.34	28.9	0.023	2.5	53.8
CinCECGTorso	1420	0.0527	3.98	28.3	0.0228	11.91	84.6
Crop	24000	0.0505	0.19	22.5	0.0234	2.69	323.0
ECG5000	5000	0.0513	1.10	27.5	0.022	6.97	174.2
ECGFiveDays	884	0.0506	11.22	49.7	0.0217	21.29	94.2
ElectricDevices	16637	0.0497	0.25	20.1	0.0203	1.54	127.9
EthanolLevel	1004	0.0525	10.80	54.3	0.021	17.75	89.2
FaceAll	2250	0.0515	2.21	24.9	0.0209	12.69	142.8
FacesUCR	2250	0.0521	2.30	25.9	0.0212	12.56	141.3
FiftyWords	905	0.0512	8.77	39.7	0.0286	22.63	102.5
FordA	4921	0.0512	0.79	19.5	0.0203	5.97	146.9
FordB	4446	0.0517	0.80	18.0	0.0205	7.62	169.4
FreezerRegularTrain	3000	0.0523	1.85	27.8	0.0229	2.88	43.2
FreezerSmallTrain	2878	0.0695	1.86	26.8	0.0215	2.82	40.7
HandOutlines	1370	0.0535	28.10	192.6	0.02	33.26	228.0
InsectWingbeatSound	2200	0.0514	2.40	26.4	0.0281	10.57	116.3
ItalyPowerDemand	1096	0.0504	15.15	83.1	0.0359	19.65	107.8
Mallat	2400	0.0537	2.70	32.4	0.0236	8.09	97.1
MedicalImages	1141	0.0510	7.60	43.4	0.0214	15.31	87.4
MixedShapesRegularTrain	2925	0.0517	1.91	27.9	0.0249	8.72	127.6
MixedShapesSmallTrain	2525	0.0510	2.28	28.9	0.025	10.57	133.4
MoteStrain	1272	0.0519	8.24	52.4	0.0282	15.95	101.5
NonInvasiveFetalECGThorax1	3765	0.0504	1.48	27.9	0.0204	4.73	89.1
NonInvasiveFetalECGThorax2	3765	0.0522	1.82	34.33	0.0209	4.38	82.6
PhalangesOutlinesCorrect	2658	0.0542	2.57	34.17	0.0202	5.98	79.6
Phoneme	2110	0.0501	4.46	47.1	0.0205	7.78	82.1
ShapesAll	1200	0.0532	6.78	40.7	0.0277	15.15	90.9
SonyAIBORobotSurface2	980	0.0520	7.03	34.5	0.0361	15.94	78.2
StarLightCurves	9236	0.0513	0.49	22.5	0.0204	2.08	95.9
Strawberry	983	0.0522	17.71	87.1	0.0217	23.47	115.5
SwedishLeaf	1125	0.0527	12.38	69.7	0.0205	20.64	116.2
Symbols	1020	0.0521	9.37	47.9	0.0211	14.5	74.0
TwoLeadECG	1162	0.0508	8.47	49.2	0.0205	17.8	103.5
TwoPatterns	5000	0.0507	1.20	30.0	0.0357	5.16	128.9
Wafer	7164	0.0548	0.75	27.0	0.0211	1.39	49.9
WordSynonims	905	0.0514	8.68	39.3	0.0285	20.93	94.8
Yoga	3300	0.0525	1.57	25.9	0.0209	7.55	124.7

Tabel B.1. Voor elke van de 38 datasets werd de MSM afstandsmatrix benaderd door ACA met $\tau = 0.05$ en $\tau = 0.02$ als stopcriterium en $\Delta_{max} = 100$. Voor elke benadering wordt de relatieve fout η, het bemonsteringspercentage en de bemonsteringsfactor gegeven.

Benadering van DTW afstandsmatrix via ACA							
		ACA ($\tau = 0.05$)			ACA ($\tau = 0.02$)		
Dataset	n	η	%	$\times n$	η	%	$\times n$
CBF	930	0.051	13.5	62.8	0.032	45.27	210.8
ChlorineConcentration	4307	0.0527	3.47	74.8	0.0204	4.65	100.2
CinCECGTorso	1420	0.0523	9.95	70.7	0.0205	26.38	187.4
Crop	24000	0.0509	0.35	42.0	0.0372	4.08	489.7
ECG5000	5000	0.0507	3.46	86.4	0.0458	9.82	245.6
ECGFiveDays	884	0.0543	20.21	89.4	0.033	36.22	160.3
ElectricDevices	16637	0.054	4.14	344.4	0.0627	2.79	232.0
EthanolLevel	1004	0.058	25.29	127.1	0.0217	42.61	214.1
FaceAll	2250	0.0513	4.2	47.3	0.0381	13.89	156.3
FacesUCR	2250	0.052	4.2	47.3	0.0379	12.62	142.1
FiftyWords	905	0.051	43.99	199.3	0.02	68.64	310.9
FordA	4921	0.0508	2.88	71.0	0.048	4.61	113.4
FordB	4446	0.054	4.9	108.9	0.0522	5.82	129.4
FreezerRegularTrain	3000	0.0733	4.98	74.7	0.0315	6.0	90.0
FreezerSmallTrain	2878	0.054	5.21	75.0	0.0213	5.97	86.0
HandOutlines	1370	0.2833	60.74	416.4	0.0202	64.73	443.7
InsectWingbeatSound	2200	0.0532	29.91	329.1	0.2395	25.66	282.4
ItalyPowerDemand	1096	0.0594	30.84	169.2	0.0225	53.97	296.0
Mallat	2400	0.0607	3.17	38.0	0.0203	10.28	123.4
MedicalImages	1141	0.0565	17.2	98.2	0.0205	48.07	274.5
MixedShapesRegularTrain	2925	0.0716	11.32	165.6	0.0826	9.64	141.1
MixedShapesSmallTrain	2525	0.0649	15.24	192.5	0.0215	50.38	636.3
MoteStrain	1272	0.0515	20.31	129.3	0.0203	53.42	340.0
NonInvasiveFetalECGThorax1	3765	0.0583	4.14	78.0	0.0206	7.56	142.4
NonInvasiveFetalECGThorax2	3765	0.0703	3.91	73.6	0.0208	7.18	135.3
PhalangesOutlinesCorrect	2658	0.0507	4.92	65.5	0.02	7.82	104.0
Phoneme	2110	0.0555	17.2	181.6	0.0429	27.75	292.9
ShapesAll	1200	0.0873	22.23	133.5	0.0219	48.24	289.7
SonyAIBORobotSurface2	980	0.0501	41.11	201.6	0.0263	72.28	354.6
StarLightCurves	9236	0.053	0.5	22.9	0.0232	2.18	100.7
Strawberry	983	0.0507	53.09	261.2	0.0217	55.51	273.1
SwedishLeaf	1125	0.052	28.43	160.0	0.0201	43.44	244.6
Symbols	1020	0.0546	12.74	65.0	0.021	14.64	74.8
TwoLeadECG	1162	0.0508	20.95	121.8	0.0215	32.68	190.0
TwoPatterns	5000	0.0534	5.55	138.7	0.0526	8.68	217.0
Wafer	7164	0.0577	0.71	25.4	0.0352	1.21	43.4
WordSynonims	905	0.0517	27.49	124.5	0.0216	57.8	261.8
Yoga	3300	0.05	3.59	59.2	0.0208	13.9	229.4

Tabel B.2. Voor elke van de 38 datasets werd de DTW afstandsmatrix benaderd door ACA met $\tau = 0.05$ en $\tau = 0.02$ als stopcriterium en $\Delta_{max} = 100$. Voor elke benadering wordt de relatieve fout η, het bemonsteringspercentage en de bemonsteringsfactor gegeven.

Benadering van MSM afstandsmatrix via SOLRADM met $\varepsilon = 2.0$							
		SOLRADM ($k = 20$)			SOLRADM ($k = 50$)		
Dataset	n	η	%	$\times n$	η	%	$\times n$
CBF	930	0.0241	21.21	98.7	0.021	46.06	214.4
ChlorineConcentration	4307	0.03	5.16	111.1	0.0133	12.74	274.4
CinCECGTorso	1420	0.0384	15.03	106.8	0.0215	34.37	244.2
Crop	24000	0.0261	0.95	114.2	0.0165	2.38	285.8
ECG5000	5000	0.0314	4.57	114.2	0.0202	11.12	278.0
ECGFiveDays	884	0.0323	23.25	102.9	0.0154	50.18	222.1
ElectricDevices	16637	0.0239	1.38	114.6	0.0155	3.41	284.1
EthanolLevel	1004	0.0302	20.99	105.5	0.017	45.02	226.2
FaceAll	2250	0.0345	9.88	111.2	0.0259	22.84	257.1
FacesUCR	2250	0.0364	9.84	110.7	0.0263	23.0	258.9
FiftyWords	905	0.042	22.74	103.0	0.0284	49.05	222.2
FordA	4921	0.026	4.6	113.2	0.0223	10.87	267.4
FordB	4446	0.0268	5.06	112.6	0.0229	12.2	271.3
FreezerRegularTrain	3000	0.0155	7.43	111.4	0.0054	17.56	263.5
FreezerSmallTrain	2878	0.0116	7.63	109.9	0.0059	18.74	269.8
HandOutlines	1370	0.024	15.35	105.2	0.0148	34.72	238.0
InsectWingbeatSound	2200	0.0357	10.06	110.7	0.0258	23.37	257.2
ItalyPowerDemand	1096	0.0384	19.49	106.9	0.0247	43.16	236.8
Mallat	2400	0.0384	9.24	110.9	0.0239	21.76	261.3
MedicalImages	1141	0.0351	18.84	107.6	0.0218	41.76	238.4
MixedShapesRegularTrain	2925	0.0356	7.61	111.4	0.0239	18.05	264.1
MixedShapesSmallTrain	2525	0.036	8.9	112.5	0.0241	20.82	263.0
MoteStrain	1272	0.0386	16.7	106.3	0.0241	37.4	238.1
NonInvasiveFetalECGThorax1	3765	0.0237	5.94	111.8	0.0167	14.23	267.9
NonInvasiveFetalECGThorax2	3765	0.0257	5.97	112.3	0.0161	14.25	268.4
PhalangesOutlinesCorrect	2658	0.0304	8.22	109.2	0.0177	19.6	260.5
Phoneme	2110	0.018	9.93	104.8	0.0142	22.46	237.1
ShapesAll	1200	0.0417	17.42	104.6	0.0247	39.17	235.2
SonyAIBORobotSurface2	980	0.0492	21.11	103.5	0.0347	45.65	223.9
StarLightCurves	9236	0.0299	2.44	112.8	0.0201	6.07	280.2
Strawberry	983	0.0301	20.77	102.2	0.0156	45.53	224.0
SwedishLeaf	1125	0.0278	18.69	105.2	0.0208	40.89	230.2
Symbols	1020	0.0257	20.33	103.8	0.0148	45.16	230.5
TwoLeadECG	1162	0.0336	18.28	106.3	0.0196	40.36	234.7
TwoPatterns	5000	0.0381	4.57	114.2	0.0295	11.06	276.6
Wafer	7164	0.0237	3.25	116.6	0.0101	7.72	276.6
WordSynonims	905	0.0413	22.84	103.5	0.0288	49.53	224.4
Yoga	3300	0.0301	6.76	111.6	0.0181	16.26	268.3

Tabel B.3. Voor elke van de 38 datasets werd de MSM afstandsmatrix benaderd door SOLRADM met $\varepsilon = 2.0$. Voor elke benadering wordt de relatieve fout η, het bemonsteringspercentage en de bemonsteringsfactor gegeven.

Benadering van DTW afstandsmatrix via SOLRADM met $\varepsilon = 2.0$							
		SOLRADM ($k = 20$)			SOLRADM ($k = 50$)		
Dataset	n	η	%	$\times n$	η	%	$\times n$
CBF	930	0.0335	22.08	102.8	0.026	47.79	222.5
ChlorineConcentration	4307	0.0189	5.11	110.1	0.0102	12.61	271.5
CinCECGTorso	1420	0.0279	15.16	107.7	0.0156	34.38	244.2
Crop	24000	0.034	0.96	115.2	0.0204	2.39	286.3
ECG5000	5000	0.0329	4.45	111.3	0.0193	10.93	273.3
ECGFiveDays	884	0.0346	23.55	104.2	0.016	50.99	225.7
ElectricDevices	16637	0.0429	1.37	114.1	0.0272	3.43	285.1
EthanolLevel	1004	0.0316	20.63	103.7	0.0166	45.69	229.6
FaceAll	2250	0.0359	9.8	110.2	0.0244	22.88	257.6
FacesUCR	2250	0.0356	9.8	110.2	0.0244	22.88	257.6
FiftyWords	905	0.0636	23.24	105.3	0.0387	49.14	222.6
FordA	4921	0.0434	4.52	111.3	0.0341	11.21	276.0
FordB	4446	0.0444	5.06	112.6	0.035	12.33	274.2
FreezerRegularTrain	3000	0.0103	7.49	112.4	0.0039	17.22	258.5
FreezerSmallTrain	2878	0.0102	7.7	110.8	0.004	17.76	255.7
HandOutlines	1370	0.031	15.22	104.3	0.0155	34.6	237.2
InsectWingbeatSound	2200	0.0716	10.14	111.6	0.0395	23.93	263.4
ItalyPowerDemand	1096	0.0285	19.16	105.1	0.0184	42.11	231.0
Mallat	2400	0.0239	9.28	111.4	0.0147	21.91	263.1
MedicalImages	1141	0.0373	19.08	108.9	0.0221	41.48	236.9
MixedShapesRegularTrain	2925	0.0491	7.51	109.9	0.0313	18.11	265.0
MixedShapesSmallTrain	2525	0.0507	8.75	110.5	0.0311	20.89	263.9
MoteStrain	1272	0.0408	16.7	106.3	0.0256	37.72	240.1
NonInvasiveFetalECGThorax1	3765	0.0193	5.97	112.3	0.0102	14.3	269.3
NonInvasiveFetalECGThorax2	3765	0.0183	5.94	111.8	0.0098	14.3	269.3
PhalangesOutlinesCorrect	2658	0.0194	8.29	110.2	0.012	19.63	261.0
Phoneme	2110	0.0378	10.43	110.1	0.0264	24.63	260.0
ShapesAll	1200	0.0504	17.81	106.9	0.0273	39.44	236.8
SonyAIBORobotSurface2	980	0.0488	21.2	104.0	0.0336	46.33	227.3
StarLightCurves	9236	0.0117	2.5	115.3	0.0073	6.06	279.7
Strawberry	983	0.0192	21.05	103.6	0.0107	46.14	227.0
SwedishLeaf	1125	0.0265	18.44	103.8	0.017	40.61	228.6
Symbols	1020	0.0093	20.42	104.2	0.0046	45.38	231.7
TwoLeadECG	1162	0.0248	18.12	105.4	0.0143	40.22	233.9
TwoPatterns	5000	0.0411	4.59	114.7	0.0252	10.95	273.7
Wafer	7164	0.0134	3.2	114.6	0.0069	7.72	276.6
WordSynonims	905	0.0446	22.84	103.5	0.0264	49.61	224.8
Yoga	3300	0.0279	6.73	111.1	0.0148	16.09	265.6

Tabel B.4. Voor elke van de 38 datasets werd de DTW afstandsmatrix benaderd door SOLRADM met $\varepsilon = 2.0$. Voor elke benadering wordt de relatieve fout η, het bemonsteringspercentage en de bemonsteringsfactor gegeven.

Bibliografie

[1] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, "The ucr time series classification archive," October 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[2] W. Meert and T. V. Craenendonck, "Time series distances: Dynamic time warping (dtw)." https://doi.org/10.5281/zenodo.3981067, 2020.

[3] E. Schubert, J. Sander, M. Ester, H. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[4] F. Petitjean, A. Ketterlin, and P. Ganarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[5] A. Stefan, V. Athitsos, and G. Das, "The move-split-merge metric for time series," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1425–1438, 2013.

[6] I. Dropbox, "Dropbox-abonnement prijzen." https://www.dropbox.com/business/plans-comparison, 2020. [Online; laatst geraadpleegd 10 augustus 2020].

[7] S. Aghabozorgi, A. Seyed Shirkhorshidi, and T. Ying Wah, "Time-series clustering a decade review," *Information systems (Oxford)*, vol. 53, pp. 16–38, 2015.

[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD96, p. 226231, AAAI Press, 1996.

[9] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[10] M. Pede, "Fast-time-series-clustering." https://github.com/MathiasPede/Fast-Time-Series-Clustering, 2020.

[11] K. A. Foster, A. M. Havenner, and A. M. Walburger, "System theoretic time-series forecasts of weekly live cattle prices," *American journal of agricultural economics*, vol. 77, no. 4, pp. 1012–1023, 1995.

[12] B. B. Nair, P. S. Kumar, N. Sakthivel, and U. Vipin, "Clustering stock price time series data to generate stock trading recommendations: An empirical study," *Expert Systems With Applications*, vol. 70, pp. 20–36, 2017.

[13] M. Ilbeigi, B. Ashuri, and A. Joukar, "Time-series analysis for forecasting asphalt-cement price," *Journal of Management in Engineering*, vol. 33, no. 1, p. 4016030, 2017.

[14] T. Kim, Y. Park, J. Myung, and E. Han, "Food price trends in south korea through time series analysis," *Public health (London)*, vol. 165, pp. 67–73, 2018.

[15] A. Wismller, O. Lange, D. R. Dersch, G. L. Leinsinger, K. Hahn, B. Ptz, and D. Auer, "Cluster analysis of biomedical image time-series," *International Journal of Computer Vision*, vol. 46, no. 2, pp. 103–128, 2002.

[16] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *Proceedings of the Seventh International Congress on Acoustics, Budapest*, vol. 3, (Budapest), pp. 65–69, Akadémiai Kiadó, 1971.

[17] C. W. Tan, M. Herrmann, G. Forestier, G. Webb, and F. Petitjean, "Efficient search of the best warping window for dynamic time warping," 01 2018.

[18] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent data analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[19] W. R. Fox, L. Kaufman, and P. J. Rousseeuw, "Finding groups in data: An introduction to cluster analysis," *Applied Statistics*, vol. 40, no. 3, pp. 486–487, 1991.

[20] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, pp. 30–34, 01 1973.

[21] D. Defays, "An efficient algorithm for a complete link method," *Comput. J.*, vol. 20, pp. 364–366, 1977.

[22] Z. Chen and Y. F. Li, "Anomaly detection based on enhanced dbscan algorithm," *Procedia Engineering*, vol. 15, pp. 178–182, 2011.

[23] M. Huang, Q. Bao, Y. Zhang, and W. Feng, "A hybrid algorithm for forecasting financial time series data based on dbscan and svr," *Information (Basel)*, vol. 10, no. 3, p. 103, 2019.

[24] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[25] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[26] M. Bebendorf, "Approximation of boundary element matrices," *Numerische Mathematik*, vol. 86, no. 4, pp. 565–589, 2000.

[27] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews. Computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[28] Q. Lei, J. Yi, R. Vaculn, L. Wu, and I. Dhillon, "Similarity preserving representation learning for time series clustering," pp. 2845–2851, 08 2019.

[29] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, pp. 43–49, February 1978.

[30] L. Gupta, D. Molfese, R. Tammana, and P. Simos, "Nonlinear alignment and averaging for estimating the evoked potential," *IEEE Transactions on Biomedical Engineering*, vol. 43, no. 4, pp. 348–356, 1996.

[31] V. Niennattrakul and C. Ratanamahatana, "Shape averaging under time warping," vol. 2, pp. 626–629, IEEE, 2009.

[32] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm," *Knowledge and information systems*, vol. 47, no. 1, pp. 1–26, 2015.

[33] E. Vidal, F. Casacuberta, J.-M. Benedí, M.-J. Lloret, and H. R. Segovia, "On the verification of triangle inequality by dynamic time-warping dissimilarity measures," *Speech Commun.*, vol. 7, pp. 67–79, 1988.

[34] F. Casacuberta, E. Vidal, and H. Rulot, "On the metric properties of dynamic time warping," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 11, pp. 1631–1633, 1987.

[35] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern Recognition*, vol. 42, pp. 2169–2180, 11 2008.

[36] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB 04, p. 792803, VLDB Endowment, 2004.

[37] I. S. Dhillon and S. Sra, "Triangle fixing algorithms for the metric nearness problem," in *Neural Information Processing Systems (NIPS)*, dec 2004.

[38] S. Goreinov, E. Tyrtyshnikov, and N. Zamarashkin, "A theory of pseudoskeleton approximations," *Linear algebra and its applications*, vol. 261, no. 1-3, pp. 1–21, 1997.

[39] A. ivril and M. Magdon-Ismail, "On selecting a maximum volume sub-matrix of a matrix and related problems," *Theoretical computer science*, vol. 410, no. 47-49, pp. 4801–4811, 2009.

[40] S. Goreinov and E. Tyrtyshnikov, "The maximal-volume concept in approximation by low-rank matrices," *Contemporary Mathematics*, vol. 208, 01 2001.

[41] S. A. Goreinov, N. L. Zamarashkin, and E. E. Tyrtyshnikov, "Pseudo-skeleton approximations by matrices of maximal volume," *Mathematical Notes*, vol. 62, no. 4, pp. 515–519, 1997.

[42] A. Cortinovis, D. Kressner, and S. Massei, "On maximum volume submatrices and cross approximation for symmetric semidefinite and diagonally dominant matrices," *Linear algebra and its applications*, vol. 593, pp. 251–268, 2020.

[43] A. Aldroubi, K. Hamm, A. B. Koku, and A. Sekmen, "Cur decompositions, similarity matrices, and subspace clustering," *Frontiers in applied mathematics and statistics*, vol. 4, 2019.

[44] A. Osinsky and N. Zamarashkin, "Pseudo-skeleton approximations with better accuracy estimates," *Linear algebra and its applications*, vol. 537, pp. 221–249, 2018.

[45] M. Bebendorf and R. Grzhibovskis, "Accelerating galerkin bem for linear elasticity using adaptive cross approximation," *Mathematical Methods in the Applied Sciences*, vol. 29, no. 14, pp. 1721–1747, 2006.

[46] T. Mach, L. Reichel, M. Van Barel, and R. Vandebril, "Adaptive cross approximation for ill-posed problems," *Journal of Computational and Applied Mathematics*, vol. 303, pp. 206–217, 2016.

[47] A. Heldring, E. Ubeda, and J. M. Rius, "Stochastic estimation of the frobenius norm in the aca convergence criterion," *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 3, pp. 1155–1158, 2015.

[48] A. Bakshi and D. P. Woodruff, "Sublinear time low-rank approximation of distance matrices," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS18, (Red Hook, NY, USA), p. 37863796, Curran Associates Inc., 2018.

[49] P. Indyk, A. Vakilian, T. Wagner, and D. P. Woodruff, "Sample-optimal low-rank approximation of distance matrices," in *Proceedings of the Thirty-Second Conference on Learning Theory* (A. Beygelzimer and D. Hsu, eds.), vol. 99 of *Proceedings of Machine Learning Research*, (Phoenix, USA), pp. 1723–1751, PMLR, 25–28 Jun 2019.

[50] A. Frieze, R. Kannan, and S. Vempala, "Fast monte-carlo algorithms for finding low-rank approximations," *Journal of the ACM (JACM)*, vol. 51, no. 6, pp. 1025–1041, 2004.

[51] X. Chen and E. Price, "Active regression via linear-sample sparsification," in *Proceedings of the Thirty-Second Conference on Learning Theory* (A. Beygelzimer and D. Hsu, eds.), vol. 99 of *Proceedings of Machine Learning Research*, (Phoenix, USA), pp. 663–695, PMLR, 25–28 Jun 2019.