SCHEIDEN VAN HANDGESCHREVEN TEKSTREGELS

Bachelorproject, 20 juli 2008

Job Zondag, Rijksuniversiteit Groningen, s1542516, j.zondag@student.rug.nl

Begeleider: Axel Brink

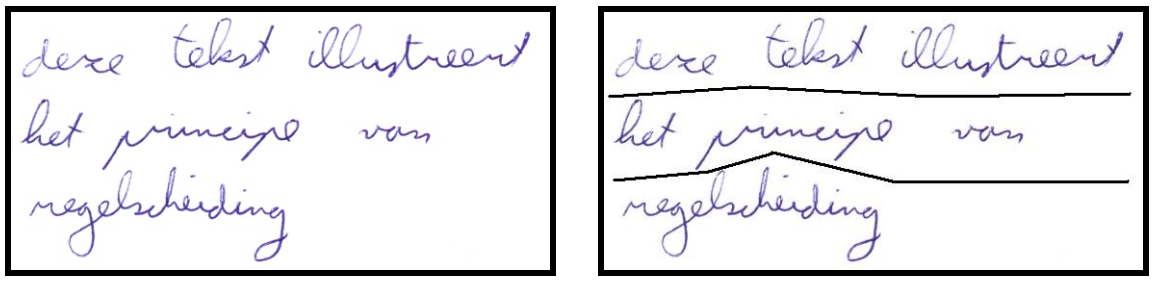
Samenvatting: Een belangrijke stap in automatische handschriftherkenning is het opdelen van geschreven documenten in losse tekstregels. Geen enkel bestaand algoritme kan echter goed omgaan met slordig geschreven tekst. In dit artikel wordt besproken waarom huidige algoritmes niet goed werken en of het mogelijk is betere algoritmes te maken. Een van de beste huidige algoritmes werkt op basis van projectiehistogrammen. In dit artikel wordt aangetoond dat dit algoritme kan worden verbeterd met twee eenvoudige aanpassingen. Het algoritme behoudt echter het fundamentele probleem dat een schatting moet worden gemaakt van de schaal van het te verwerken document. Een alternatieve methode op basis van een Kohonen self-organizing map (KSOM) en een methode op basis van een genetisch algoritme worden geïntroduceerd. Deze algoritmes vormen geen betere oplossing dan het algoritme op basis van projectiehistogrammen. Uit een evaluatie van deze vier algoritmes blijkt dat het voor automatische regelscheiding van belang is meerdere kenmerken van tekstregels te benutten. Het integreren van deze verschillende kenmerken in een enkel systeem is problematisch.

1. Inleiding

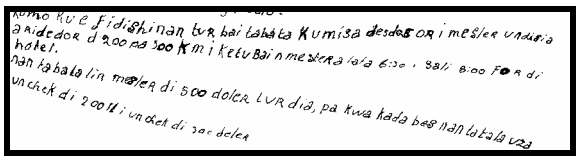
Voor het uitvoeren van automatische handschriftherkenning is het noodzakelijk dat de regels in een tekst eerst van elkaar worden gescheiden. Dit moet worden gedaan omdat hierna pas woordsegmentatie en woordherkenning kan worden toegepast. Daarom is een algoritme nodig dat voor elk stukje inkt bepaalt tot welke regel het behoort. Een regelscheidingsalgoritme geeft dus aan waar de regelovergangen in een tekst zich bevinden. Dit principe is geïllustreerd in figuur 1.1.

Voor mensen is het opdelen van een tekst in afzonderlijke tekstregels een eenvoudige taak. Zelfs in tekst die we niet kunnen lezen, omdat deze bijvoorbeeld slordig is of in een onbekende taal is geschreven, kunnen we doorgaans zonder problemen zien waar de regels zich bevinden en kunnen we lijnen trekken tussen de regels. Dit doet vermoeden dat regelscheiding automatisch kan worden uitgevoerd. Er zijn een aantal algoritmes ontwikkeld die dit kunnen, maar nog geen enkel algoritme kan goed omgaan met slordig geschreven tekst. Uit data van het NFI (Nederlands Forensisch Instituut) blijkt verder dat tekstregels kunnen divergeren (figuur 1.2), krom zijn en soms aan het einde van de pagina naar beneden afbuigen (figuur 1.3). Veel regelscheidingsalgoritmes hebben moeite met dergelijke tekst.

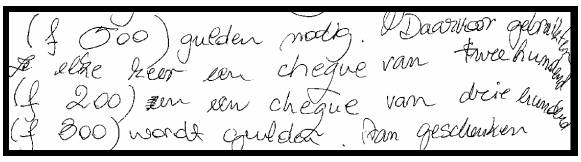
In dit artikel wordt onderzocht waarom het zo moeilijk is automatisch tekstregels van elkaar te scheiden en er worden alternatieve methoden voor regelscheiding voorgesteld. Hierbij ligt de nadruk op grove regelscheiding. Algoritmes



Figuur 1.1: Regelscheiding betekent het aangeven waar de regelovergangen zich bevinden in een tekst.



Figuur 1.2: Divergerende tekstregels.*



Figuur 1.3: Afbuigende tekstregels.*

moeten kunnen aangeven waar de regel-overgangen zich bevinden, maar hoeven niet te bepalen bij welke tekstregel pixels behoren die dicht bij de regelovergang liggen.

1.1. Eerder onderzoek

Een manier om regels van elkaar te scheiden is voorgesteld door Nicolas et al. (2004). In het artikel wordt regelscheiding voorgesteld als een zoekprobleem. Het gehele document bestaat uit een verzameling *connected components* (verbonden componenten). Een enkele regel wordt gezien als een aantal bij elkaar horende connected components. Het algoritme bestaat uit twee productieregels die een nieuwe regel aan de oplossing kunnen toevoegen of een component kunnen toevoegen aan een regel in de oplossing. Dit wordt gedaan tot er geen losse componenten meer over zijn. Het toevoegen wordt geleid door een aantal kostenregels. Aan elke productieregel zijn kosten verbonden die afhangen van de waarschijnlijkheid dat een productieregel moet wordt toegepast. Deze kans wordt bepaald door de afstand tussen de componenten. Hoe groter deze afstand is hoe kleiner de kans dat de componenten met elkaar moeten worden verbonden en des te hoger zijn daarom de kosten. De methode maakt gebruik van het A*-algoritme om zo snel mogelijk de oplossing met de laagste kosten te vinden.

In het beschreven algoritme wordt alleen gebruikt gemaakt van de afstand tussen de componenten. Dit blijkt problematisch te zijn als de regels dicht op elkaar geschreven zijn. De auteurs stellen daarom voor ook andere tekstuele kenmerken, zoals de continuïteit van regels, toe te voegen. Deze toevoegingen kunnen worden gebruikt bij het berekenen van de kosten van het toepassen van een productieregel. Een tweede probleem van deze benadering is dat een connected component maar aan één regel toegekend kan worden. Als regels overlappen kan een connected component soms elementen uit verschillende regels bevatten. Dit algoritme kan hier niet mee omgaan.

Een ander algoritme, dat wel om kan gaan met overlap maakt gebruik van het horizontale projectiehistogram van het handschrift. Een horizontaal projectiehistogram wordt berekend door de donkere pixels in een document in horizontale richting op te tellen. De regel-overgangen liggen ter hoogte van de dalen in het histogram (Likforman-Sulem *et al.*, 2006).

Het probleem van een dergelijke benadering is dat deze niet overweg kan met hand-geschreven tekstregels die niet horizontaal en recht zijn. Als de regels recht zijn en parallel aan elkaar lopen is schuinheid van regels nog te corrigeren door de projectieas te draaien tot deze haaks op de schrijfrichting staat (Shapiro *et al.*, 1993). In de praktijk blijkt echter dat, als op ongelinieerd papier is geschreven, regels vaak niet recht en parallel zijn.

Een beter algoritme wordt gegeven door Arivazhagan et al. (2007). Zij beschrijven een algoritme dat het document opdeelt in een twintigtal verticale stroken. Op elke strook wordt de hierboven beschreven projectie-methode toepast. Vervolgens worden de dalen in de aanliggende stroken met elkaar verbonden. Op deze manier kan worden omgegaan met schuine regels.

Deze methode maakt bij het verwerken van schuine tekstregels echter wel fouten. Problemen ontstaan wanneer de stroken te breed zijn, omdat dan bij schuine tekstregels de projectie-histogrammen vervlakken. Het wordt dan moeilijker pieken en dalen te onderscheiden. Aan de andere kant mogen de stroken ook niet te smal worden aangezien er dan niet genoeg pixels zijn om duidelijke pieken en dalen te vinden. Het lijkt er dus op dat ook deze methode niet perfect is en dat het automatisch scheiden van tekstregels een probleem is dat nog niet volledig is opgelost.

1.2. Onderzochte algoritmes

Om te onderzoeken waarom het zo moeilijk is regels automatisch van elkaar te scheiden is een viertal algoritmes geïmplementeerd. Een eerste algoritme (algoritme 1) is gemaakt op basis van het algoritme van Arivazhagan et al. (2007). Het tweede algoritme is een aanpassing van dit algoritme (algoritme 2). Het doel van deze aanpassing is te kijken of het zinvol is om het algoritme te blijven optimaliseren, of dat vervolgonderzoek beter kan worden gericht op het zoeken naar nieuwe benaderingen.

De informatie die door algoritme 1 en 2 wordt uitgebuit is de dichtheid van donkere pixels. Binnen tekstregels zijn meer donkere pixels te vinden dan in de regelovergangen. Dit

doet vermoeden dat het zinvol is het probleem te benaderen als een clusteringprobleem. Algoritme 3 traint daarom een (aangepaste versie van een) Kohonen neuraal netwerk op de pagina met tekst. Het is de bedoeling dat elke rij neuronen precies op één tekstregel komt te liggen. De lijnenstukken die de regels van elkaar scheiden kunnen precies tussen de rijen neuronen worden getrokken.

Naast de dichtheid van donkere pixels zijn er nog andere kenmerken toe te kennen aan tekstregels. Regels zijn namelijk continu (woorden liggen in elkaars verlengde) en de afstand tussen de regels is bij benadering constant. Algoritme 4 gebruikt deze drie kenmerken als elementen in een kostenfunctie van een genetisch algoritme, om een evolutionair zoekproces naar de optimale regelscheiding te leiden.

Ten slotte zijn de geïmplementeerde algoritmes met elkaar vergeleken om te kijken welke het beste werkt. Een vergelijking is gemaakt zowel op basis van visuele impressies, als op basis van een kwantitatieve test.

2. Methode

De methode om regels te scheiden zoals wordt beschreven door Arivazhagan et al. (2007), is gebaseerd op projectiehistogrammen. Projectiehistogrammen beschrijven de verdeling van donkere pixels in een (deel van een) afbeelding. Sectie 2.1 beschrijft hoe projectiehistogrammen worden gegenereerd en in sectie 2.2 wordt beschreven hoe deze histogrammen worden gebruikt om tekstregels van elkaar te scheiden (algoritme 1). Sectie 2.3 beschrijft een aangepaste versie van dit algoritme (algoritme 2). De methode die regels scheidt op basis van een Kohonen self-organizing map (algoritme 3) wordt beschreven in sectie 2.4. Sectie 2.5 beschrijft het algoritme op basis van een genetisch algoritme (algoritme 4). Sectie 2.6 beschrijft de kwantitatieve test waarmee de algoritmes worden beoordeeld.

2.1. Projectiehistogrammen

Een horizontaal projectiehistogram wordt berekend door binnen een afbeelding of een deel van die afbeelding het aantal donkere pixels in horizontale richting op te tellen. Dit resulteert in een lijst getallen, voor elke rij pixels één. Er van uitgaande dat de tekst met donkere inkt op licht gekleurde ondergrond geschreven is, bevinden zich op plaatsen waar een regel is geschreven veel donkere pixels, en op plaatsen waar zich een regelovergang bevindt weinig. In het projectiehistogram resulteert een regel in een top en een regelovergang in een dal.

Omdat in projectiehistogrammen kan worden gezien waar zich (waarschijnlijk) regelovergangen bevinden kunnen zij worden gebruikt voor het opdelen van een document in afzonderlijke tekstregels. Wat gedaan moet worden is het zoeken naar de minima die de regelovergangen representeren. De representaties van de regelovergangen worden vervolgens ter hoogte van deze minima in horizontale richting gedefinieerd.

In de praktijk blijkt dat als een projectiehistogram wordt gemaakt van een stuk tekst, er veel locale maxima en minima in het histogram zitten. Dit is een probleem aangezien in dit geval meer regelovergangen worden gevonden dan dat er daadwerkelijk in het document aanwezig zijn. Het is van belang alleen de minima te vinden die daadwerkelijk een regelovergang representeren. De locale minima en maxima moeten daarom weg worden gefilterd door het histogram glad te maken. Hiervoor gebruiken Arivazhagan et al. (2007) een *n-average filter* (middelingsfilter). Dit filter bleek echter niet voldoende om alle locale minima te verwijderen. Daarom is nog een tweede criterium opgenomen om te bepalen of een minimum als regelovergang moet worden beschouwd of niet.

Het n-average filter berekent een nieuwe waarde voor elk punt in het projectiehistogram door het gemiddelde te nemen van de betreffende waarde en de n voorafgaande en opvolgende waarden. De verschillen tussen aanliggende waarden in het projectiehistogram worden daardoor minder groot. Het gevolg is dat het histogram gladder wordt. De breedte van het filter is sterk van invloed op de werking van het algoritme. Voor deze implementatie is gekozen voor een n van 1 procent van de hoogte van het document omdat algoritme 1 met deze filterbreedte het minste fouten maakt. In de resultatensectie zal deze keuze nader worden toegelicht.

Na het toepassen van het n-average filter is een groot deel van de locale minima en maxima verdwenen. Om te zorgen dat de gevonden

Algoritme 2.1: Verwijder locale minima en maxima

```
1   VERWIJDERLOCALEWAARDEN ( List, Treshold )
2   % List is de lijst met alle minima en maxima uit
    het histogram
3   % Treshold is de drempelwaarde
4   % opeenvolgende maxima en minima met een
    % verschil in waarde dat onder de treshold ligt
    % worden verwijderd
5   start←1, end ← ||List|| - 2
6   while start ≤ end
7       ab ← | List[start] - List[start+1] |
8       bc ← | List[start+1] - List[start+2] |
9       if ab < Treshold or bc < Treshold then
10          if ab < bc
11              REMOVE(List[start])
12              REMOVE(List[start])
13              if start > 1 then
14                  Start ← start – 1
15              end if
16          Else
17              REMOVE(List[start+1])
18              REMOVE(List[start+1])
19          end if
20          end ← end - 2
21      else
22          start ← start + 1
23      end if
24  end while
```

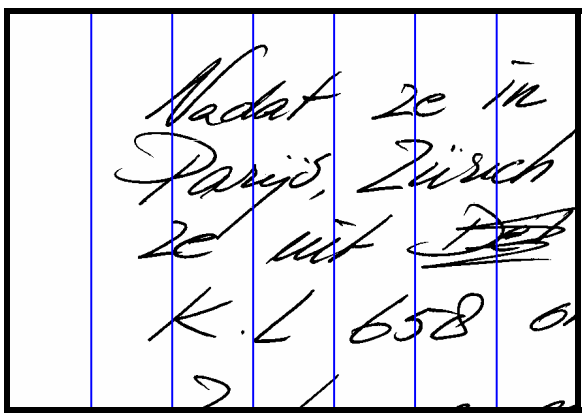
minima en maxima daadwerkelijke regelovergangen en tekstregels representeren worden opeenvolgende minima en maxima die een waarde hebben die dicht bij elkaar liggen verwijderd. De drempelwaarde die wordt gebruikt om te bepalen of het verschil tussen een maximum en minimum te klein is, is gedefinieerd als een kwart van het maximale verschil tussen een direct op elkaar volgend minimum en maximum in het histogram. Op deze manier worden alleen maxima en minima waartussen het verschil redelijk groot is bewaard. Hierbij wordt er van uit gegaan dat er geen groot verschil bestaat tussen de lengte van de regels binnen het gebied waarvan een projectiehistogram wordt gemaakt. Als dit wel het geval is bestaat de kans dat korte regels een te kleine top vormen in het histogram en als een locaal maximum worden gezien.

Locale minima en maxima kunnen op een lineaire manier worden verwijderd met behulp van algoritme 2.1. Het algoritme gaat alle minima en maxima in het histogram langs en verwijdert alle paren van een direct op elkaar volgend minimum en maximum waartussen het verschil onder de drempelwaarde ligt.

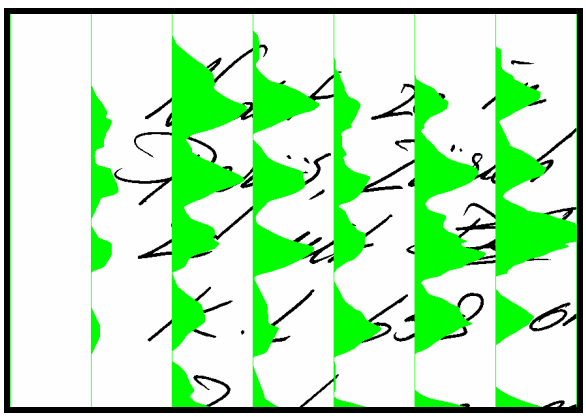
2.2. Algoritme 1: Regelscheiding op basis van projectiehistogrammen

Arivazhagan et al. (2007) onderscheiden in het algoritme om tekstregels te scheiden twee stappen. In de eerste stap wordt op basis van projectiehistogrammen bepaald waar ongeveer de regelovergangen zich bevinden. De tweede stap gebruikt deze uitkomst om de precieze regelovergangen te bepalen door gebruik te maken van statistische methoden. Aangezien het in dit artikel gaat om het grofweg scheiden van tekstregels, wordt door dit artikel alleen het eerste deel van het algoritme besproken.

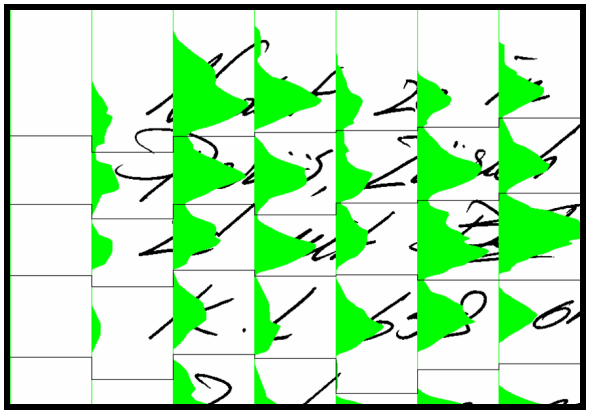
Het algoritme deelt het document op in een twintigtal verticale stroken van gelijke breedte (figuur 2.1). Voor elk van deze stroken wordt het projectiehistogram berekend (figuur 2.2.) en wordt bepaald waar zich de dalen van het projectiehistogram bevinden. Omdat deze stroken niet erg breed zijn is er weinig kans dat het projectiehistogram teveel wordt afgevlakt bij schuine tekstregels. De dalen worden daarom gezien als een betrouwbare schatting van de plaats waar zich, binnen de strook, de regelovergang bevindt. Vervolgens worden de



Figuur 2.1: Een deel van een afbeelding die is opgedeeld in verticale stroken.*



Figuur 2.2: Van elk van de stroken wordt een projectiehistogram gemaakt.*



Figuur 2.3: De minima in de projectiehistogrammen worden met elkaar verbonden. *

minima die behoren tot dezelfde regelovergang met elkaar verbonden (figuur 2.3).

De manier waarop de minima uit de aanliggende stroken met elkaar moeten worden verbonden is niet triviaal, aangezien het aantal minima tussen opeenvolgende stroken kan verschillen. Dit gebeurt als een bepaalde regel niet zo lang is als de andere regels, als een regel pas halverwege het document begint of als de regelovergang niet is gevonden in het projectie-histogram.

Om de juiste verbindingen tussen de minima te maken worden de stroken van links naar rechts één voor een verwerkt. Voor elke strook wordt elk minimum verbonden met het minimum in de vorige strook dat hier het dichtst bij in de buurt ligt. Als er meerdere minima met een zelfde minimum in de vorige strook zouden worden verbonden, blijft alleen de verbinding met het dichtstbijzijnde minimum behouden. De andere minima worden gezien als de introductie van nieuwe regelovergangen. Vervolgens wordt voor elk met elkaar verbonden paar gekeken of hun afstand een drempelwaarde overschrijdt. Als dit het geval is wordt de verbinding verbroken en wordt het minimum in de rechter strook gezien als de introductie van een nieuwe regelovergang. Als dit niet het geval is blijven de minima van de twee horizontale lijnstukken daadwerkelijk met elkaar verbonden. Dit zijn dus twee opeenvolgende elementen van een enkele representatie van een regelovergang. Als een minimum niet verbonden is met een minimum in de volgende strook wordt de representatie van de regelovergang in horizontale richting doorgetrokken. De drempel-waarde die wordt gebruikt om te bepalen of twee lijnstukken met elkaar moeten worden verbonden, is de gemiddelde hoogte van de tekstregels.

Omdat in veel documenten de tekstregels niet precies beginnen aan de linker rand, maar een zekere kantlijn open laten, is het niet verstandig de representatie van de regel-overgangen te laten beginnen met de minima uit het projectiehistogram van de eerste (meest linkse) strook. Daarom wordt voor de begin-punten van de scheidingslijnen de minima uit het projectiehistogram van de eerste 25% van de linkerkant van het document gebruikt.

2.3. Algoritme 2: Aanpassing algoritme 1

Algoritme 1 kan maakt vaak fouten bij schuine tekstregels en het algoritme detecteert regel-overgangen soms niet. Om tegemoet te komen aan deze fouten zijn voor dit algoritme een tweetal aanpassingen gemaakt. In de eerste plaats is het n-average filter dat gebruikt werd voor het verwijderen van locale minima en maxima in het projectiehistogram vervangen door een gaussiaans filter. Het n-average filter maakt van elke waarde in het projectiehistogram het gemiddelde van een n aantal aanliggende waarden. Dit betekent dat de waarden voor en na het betreffende punt even veel invloed hebben op de uiteindelijke waarde. Met een gaussiaans filter is de wegingsfactor die aan de aanliggende waarden wordt toegekend meer evenwichtig. Waarden die dichtbij het betreffende punt liggen wegen zwaarder mee dan waarden die verder van het betreffende punt af liggen. De sigmawaarde voor het filter is vastgesteld op 0.5 % van de hoogte van het document. Deze keuze wordt verder toegelicht in de resultatensectie.

Een tweede aanpassing bestaat er uit de stroken voor een deel te laten overlappen. Er worden twee maal zo veel stroken gebruikt. De stroken behouden de breedte van vijf procent van de breedte van het document, alleen is de beginpositie van elke strook nu 2.5 % van de paginabreedte opgeschoven ten opzichte van de vorige strook in plaats van vijf procent. De stroken overlappen elkaar dus voor vijftig procent. Dit betekent dat een representatie van een regelovergang uit twee maal zoveel opeenvolgende minima bestaat als in het oorspronkelijke algoritme.

Het voordeel ten opzichte van het oorspronkelijke algoritme is dat de minima die met elkaar moeten worden verbonden nu dichter bij elkaar liggen. Dit betekent dat bij schuine tekstregels het verschil in hoogte van de regelovergang tussen twee opeenvolgende stroken minder groot is. Hierdoor is de kans dat twee minima foutief met elkaar worden verbonden minder groot. Dit maakt het aangepaste algoritme beter in staat om te gaan met schuine tekst.

2.4. Algoritme 3: KSOM

Algoritme 1 en 2 maken gebruik van de kennis dat zich in tekstregels relatief veel donkere pixels (waaruit de tekst is opgebouwd) bevinden en in de regelovergangen juist weinig. Dit doet vermoeden dat regelscheiding kan worden gezien als een clusteringprobleem. Daarom is een algoritme ontworpen dat regelscheiding kan uitvoeren op basis van een *Kohonen self-organizing map* (KSOM). In dit algoritme krijgt het netwerk in de trainingsfase de coördinaten aangeboden van willekeurige donkere pixels in de pagina. Het doel van de trainingsfase is dat op elke tekstregel een rij neuronen komt te liggen. Na de trainingsfase worden de regelovergangen bepaald door het midden te nemen tussen de rijen neuronen die de regels representeren.

Het netwerk bestaat uit neuronen die twee waarden bevatten, een x- en een y-positie. De neuronen zijn in horizontale en verticale richting met elkaar verbonden. Een neuron is alleen verbonden met zijn directe buren, elk neuron (met uitzondering van de neuronen aan de rand van het netwerk) heeft dus een verbinding met vier andere neuronen. Het netwerk wordt geïnitialiseerd met twintig neuronen per rij. Het aantal rijen neuronen moet gelijk zijn aan het aantal tekstregels in het document. Dit aantal wordt geschat door het document te laten verwerken door algoritme 2. Het met dit algoritme gevonden aantal regelovergangen plus één is het aantal verwachte aanwezige regels. Het aantal rijen neuronen is gelijk aan deze waarde. De neuronen zijn in de x-richting gelijkmatig over de breedte van de afbeelding verdeeld. De initiële y-positie van de neuronen wordt willekeurig gekozen rond het midden van de afbeelding.

Na de initialisatie volgen 10.000 tijdstappen waarin het netwerk zich aanpast aan de afbeelding. Het netwerk ontvouwt zich zodat de lijnstukken die de neuronen in horizontale richting met elkaar verbinden elkaar niet meer kruisen en de rijen neuronen elk op een enkele tekstregel komen te liggen. De volgorde van de rijen neuronen is gelijk aan de volgorde van de regels in het document. De juiste volgorde is van belang aangezien de regelovergangen worden gevonden door het midden te nemen tussen de rijen neuronen.

Tijdens elke tijdstap wordt een willekeurige donkere pixel gekozen. Deze donkere pixel functioneert als voorbeeldelement van de tekstregel waaruit hij afkomstig is. Vervolgens wordt het dichtstbijzijnde neuron bepaald (het winnende neuron). Dit neuron wordt op dat moment beschouwd als de representatie van de cluster (een deel van een tekstregel) waarin de donkere pixel zich bevindt. Het neuron wordt in de richting van de pixel verplaatst, afhankelijk van de leersnelheid. Gegeven dat de pixel representatief is voor de cluster die het neuron beschrijft, komt het neuron op den duur steeds meer in het midden van de cluster te liggen. De neuronen die zich in de *neighborhood* (omgeving) van het winnende neuron bevinden worden ook in richting van de pixel verplaatst afhankelijk van de leersnelheid en de afstand tot het winnende neuron. De neighborhood en leersnelheid worden naarmate het leerproces vordert steeds kleiner gedefinieerd waardoor de aanpassingen steeds minder groot worden en het netwerk convergeert.

De aanpassingen die worden gemaakt betreft alleen de y-positie van de neuronen, de x-positie wordt niet aangepast. Hiervoor zijn drie redenen. In de eerste plaats zorgt deze beperking er voor dat de rijen neuronen zich over de gehele breedte van de afbeelding uitstrekken. Dit is van belang omdat op deze manier de regelovergangen beter kunnen worden aangegeven. Als een tekstregel korter is dan de daaropvolgende tekstregel kan op deze manier de representatie van de regelovergang toch over de hele lengte van de langste regel worden bepaald. Ten tweede wordt gewaarborgd dat de rijen neuronen (bij benadering) horizontaal op de tekstregels komen te liggen en niet verticaal. De individuele neuronen representeren clusters

donkere pixels en niet volledige tekstregels. Een tekstregel is een opeenvolging van clusters. Het netwerk leert in feite alleen clusters. Zonder het vastzetten van de x-positie kan het netwerk zich negentig graden draaien. De neuronen liggen dan nog steeds op de tekstregels, maar de kolommen representeren nu de regels in plaats van de rijen. Een derde reden voor het niet aanpassen van de x-posities is dat de kans kleiner is dat een enkel neuron een cluster beschrijft dat meer dan één tekstregel omvat.

De grootte van de aanpassing van de y-waarde van een neuron ($\Delta node\_y$) in een rij (i) en een kolom (j) voor een enkele tijdstap is weergegeven in vergelijking 2.1. De aanpassing is gelijk aan het verschil in y-positie tussen de pixel (pix_y) en het neuron, vermenigvuldigd met een leersnelheid (α) en een gewichtswaarde (w) die afhangt van de plaats van het neuron in de neighborhood (N).

$$\Delta node\_y_{i,j} = \left(pix\_y - node\_y_{i,j}\right) * \alpha * w_{i,j} \quad (2.1)$$

De neighborhood in de horizontale richting (N_i) wordt geïnitialiseerd op de helft van het aantal neuronen dat het netwerk breed is (x_nodes) en neemt lineair af met de tijd (de huidige tijdstap t) en is na het totaal aantal tijdstappen (n) gelijk aan 0 (vergelijking 2.2). De neighborhood in de verticale richting (N_j) wordt geïnitialiseerd op een kwart van het aantal neuronen dat het netwerk hoog is (y_nodes), neemt lineair met de tijd af en is 0 na de helft van de tijdstappen (vergelijking 2.3).

$$Ni = x\_nodes * (n - t)/2n \quad (2.2)$$

$$Nj = \begin{cases} y\_nodes * (0.5n - t)/2n & : t < n/2 \\ 0 & : t \geq n/2 \end{cases} \quad (2.3)$$

De leersnelheid (α) wordt geïnitialiseerd op 0.5 en neemt lineair af met de tijd en is gelijk aan 0 na de laatste tijdstap (n) (vergelijking 2.4). De gewichtswaarde (w) waarmee de aanpassing wordt vermenigvuldigd is 0 als het neuron niet valt binnen de verticale grootte van de neighborhood (N_j) van het winnende neuron ($nearest$) (vergelijking 2.5). Een neuron hoeft niet binnen de horizontale neighborhood (N_i) van het winnende neuron te vallen om een gewichtswaarde groter dan 0 te krijgen. Dit waarborgt de continuïteit van de regel-representaties. De horizontale neighborhood wordt wel gebruikt om de gewichtswaarden te bepalen.

$$\alpha = \frac{0.5 * (n - t)}{n} \quad (2.4)$$

$$w_{i,j} = \begin{cases} w\_x_{i,j} * w\_y_{i,j} & : |j - nearest\_j| < Nj \\ 0 & : |j - nearest\_j| \geq Nj \end{cases} \quad (2.5)$$

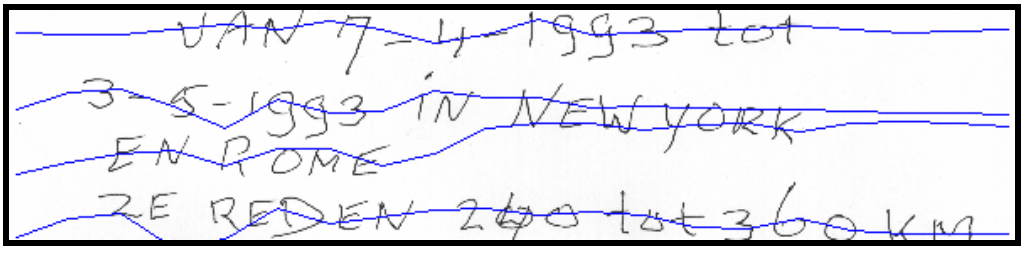
De gewichtswaarden waarmee de aanpassingen van neuronen worden vermenigvuldigd zijn weergegeven in vergelijking 2.6 en 2.7. Het zijn gaussiaanse functies, neuronen die dicht bij het winnende neuron in de buurt liggen worden met een hogere waarde vermenigvuldigd dan neuronen die hier ver van af liggen. De gaussiaanse functies zijn niet genormaliseerd (de normalisatieterm die er voor zorgt dat de integraal van de functie gelijk is aan 1 is weggelaten) om te voorkomen dat gewichtswaarden groter kunnen worden dan 1. De variabelen in de functie zijn de horizontale en verticale neighborhood (N_i en N_j) en de afstand in het netwerk tussen het neuron op positie i,j en het winnende neuron ($nearest$).

$$w\_x_{i,j} = \exp\left(\frac{-(i - nearest\_i)^2}{2 * Ni^2}\right) \quad (2.6)$$

$$w\_y_{i,j} = \exp\left(\frac{-(j - nearest\_j)^2}{2 * Nj^2}\right) \quad (2.7)$$

2.4.1. Toevoegingen aan algoritme 3

Het algoritme zoals dat hierboven is beschreven maakt veel fouten waarbij op een enkele tekstregel meerdere rijen neuronen vlak boven elkaar komen te liggen. Een voorbeeld hiervan is figuur 2.4. Daarnaast ontstaan fouten doordat de



Figuur 2.4: Twee regelrepresentaties liggen op één tekstregel. *

initiële schatting van het aantal tekstregels soms niet correct is. Om fouten te voorkomen zijn twee elementen aan het algoritme toegevoegd.

Een eerste element detecteert of er te weinig regels aanwezig zijn. Dit wordt gedaan door tussen tijdstap 4.000 en 8.000 te kijken of er (delen van) regelrepresentaties zijn die nog veel grote aanpassingen maken. Als dat het geval is ontbreekt op die plaats waarschijnlijk een regelrepresentatie waardoor de aanwezige regelrepresentatie door twee tekstregels heen en weer wordt getrokken. Op deze plaats wordt een nieuwe regelrepresentatie ingevoegd waarna het netwerk 1.000 extra tijdstappen krijgt om de twee regelrepresentaties te laten aanpassen. Een regelrepresentatie wordt hiervoor geselecteerd als zijn som van absolute aanpassingen tijdens 100 tijdstappen meer dan drie keer zo veel is als de gemiddeld aanpassing van een tekstregel gedurende die 100 tijdstappen.

Een tweede element detecteert of (delen van) regels te dicht bij elkaar liggen. Als alle punten van twee regelrepresentaties na tijdstap 4.000 te dicht bij elkaar liggen, beschrijven ze waarschijnlijk dezelfde tekstregel en moet één van de twee regels worden verwijderd. Twee regelpunten liggen te dicht bij elkaar als hun afstand minder is dan de helft van de gemiddelde regelafstand in het document.

Als slechts een deel van twee tekstregels te dicht bij elkaar liggen na tijdstap 4.000 representeren ze waarschijnlijk twee regels die niet even lang zijn. Dat betekent dat een deel van de representatie van de korte regel op de langere regel is komen te liggen. De representatie van de korte regel moet daarom worden rechtgezet. Het rechtzetten bestaat er uit dat de regelrepresentatie die de korte regel beschrijft parallel wordt gelegd aan de representatie van de lange tekstregel met een afstand die gelijk is aan de gemiddelde afstand tussen alle tekstregels. De representatie die de kortste tekstregel beschrijft wordt gevonden door de representatie te zoeken die wat betreft zijn vorm het meest afwijkt van de andere regelrepresentaties in het document.

2.5. Algoritme 4: Genetisch algoritme

Het volgende algoritme scheidt regels van elkaar met behulp van een genetisch algoritme. Het zoekproces begint met een initiële set van twintig *kandidaat-oplossingen*. Een kandidaat-oplossing is een beschrijving van alle tekstregels in het document. Van de kandidaat-oplossingen wordt met behulp van een kostenfunctie uitgerekend hoe goed zij het document beschrijven. De vijf beste oplossingen worden bewaard, de andere vijftien worden verwijderd. De vijftien vrijgekomen plaatsen worden daarna ingevuld met *mutaties* van de vijf beste oplossingen. Van deze nieuwe set kandidaat-oplossingen worden wederom de vijf beste oplossingen geselecteerd. Dit proces wordt 1.000 keer herhaald, waarbij de kandidaat-oplossingen steeds beter worden. Vervolgens wordt de beste oplossing bepaald. De representaties van de regelovergangen worden gevonden door het midden te nemen tussen de representaties van de tekstregels.

Een representatie van een tekstregel bestaat uit een twintigtal punten met een x- en een y-positie. Het aantal regelrepresentaties wordt bepaald door een schatting te maken van het aantal tekstregels door middel van algoritme 2. Het aantal gevonden regelovergangen plus één wordt gebruikt als aantal benodigde regelrepresentaties. De y-posities van de punten worden willekeurig geïnitialiseerd binnen de eerste tien procent van de hoogte van het document. Dit heeft tot gevolg dat de initiële regelrepresentaties relatief recht zijn. De punten worden horizontaal gelijkmatig verdeeld. Net als bij het KSOM algoritme zorgt dit er voor dat de regelovergangen over de gehele breedte van het document kunnen worden bepaald.

De kostenfunctie beschrijft hoe goed een kandidaat-oplossing de tekstregels in het document beschrijft. Deze functie bevat een drietal elementen (*el1*, *el2* en *el3*) die in een eerstegraads functie worden samengevoegd (vergelijking 2.8, α, β en γ zijn constanten). Deze functie wordt verder besproken in de resultaten-sectie.

$$kosten = \alpha * el1 + \beta * el2 + \gamma * el3 \qquad (2.8)$$

Het eerste element van de kostenfunctie is de som van de afstanden van alle donkere pixels tot de dichtstbijzijnde regelrepresentatie. Bij een goede oplossing liggen de regelrepresentaties op de tekstregels. Binnen de tekstregels liggen relatief veel donkere pixels. Dat betekent dat de afstand van de pixels tot de dichtstbijzijnde regelrepresentatie kleiner is als de representaties

op de tekstregels liggen, dan wanneer ze niet op de tekstregels liggen.

Het tweede element van de kostenfunctie maakt gebruik van de hoeken tussen de punten van de regelrepresentaties. Drie opeenvolgende punten in een regelrepresentatie vormen een hoek. Als een regelrepresentatie recht is zijn de hoeken binnen die representatie elk 180 graden. Elementen in een tekstregel liggen in elkaars verlengde. Dat betekent dat kan worden verwacht dat de hoek tussen drie punten niet veel afwijkt van 180 graden. Als die hoek veel groter of kleiner is, is de kans groot dat niet alle punten in een regelrepresentatie op dezelfde tekstregel liggen. De kostenfunctie bevat daarom de som van het absolute verschil tussen de hoek en 180 graden voor alle punten in een kandidaat-oplossing.

Het derde kenmerk van tekstregels dat is verwerkt in de kostenfunctie is dat de afstand tussen twee opeenvolgende tekstregels over de gehele lengte van de regels ongeveer gelijk is. Het algoritme rekent voor elk opeenvolgend paar regelrepresentaties de gemiddelde onderlinge afstand uit. Vervolgens wordt de som genomen van het verschil tussen deze gemiddelde afstand en de afstand tussen de twee tekstregels ter hoogte van de punten waaruit de regelrepresentaties bestaan. Als de twee regelrepresentaties parallel aan elkaar lopen ligt deze waarde dicht bij nul. Voor elke opeenvolgend paar regelrepresentaties wordt deze waarde uitgerekend. De som van deze waarden vormt het derde element in de kostenfunctie.

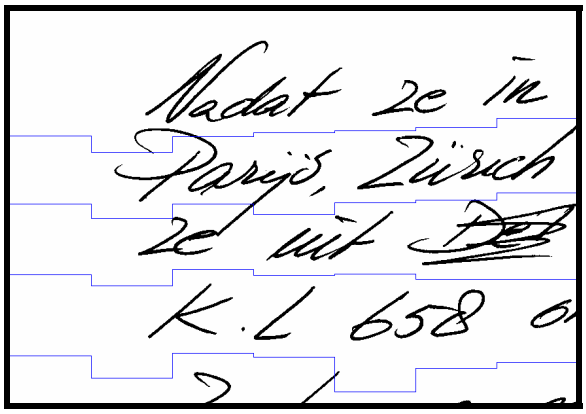
De kostenfunctie wordt gebruikt om de vijf kandidaat-oplossingen met de laagste kosten te selecteren. Dit zijn de vijf beste kandidaat-oplossingen die worden meegenomen naar de volgende tijdstap. De nieuwe kandidaat-oplossingen zijn mutaties van deze oplossingen. Tijdens een mutatie wordt één van de vijf beste oplossingen uit de vorige set kandidaat-oplossingen gekozen en een klein beetje aangepast. De aanpassing bestaat er uit dat een willekeurig punt in een willekeurige regelrepresentatie wordt geselecteerd. De y-positie van dit punt wordt vervolgens verhoogd of verlaagd met een willekeurige waarde die ligt tussen 0 en een kwart van de hoogte van de pagina. Deze aanpassing kan zowel een gunstig als een ongunstig effect hebben op de kosten van de kandidaat-oplossing. Door telkens kandidaat-oplossingen die het beste zijn te bewaren convergeert het algoritme naar een oplossing die de afbeelding goed weet te beschrijven.

2.6. Testmethode

De prestaties van de algoritmes 1 tot en met 3 zijn getest met behulp van een eenvoudige methode die fouten telt. Algoritme 4 is wegens zijn slechte prestaties niet op deze manier getest. Voor de test is een testset gebruikt bestaande uit 58 gescande handschriften uit de NFI dataset. Deze dataset werd voor het eerst geïntroduceerd in (Brink *et al.*, 2007). De set bestaat uit 3501 gescande formulieren met handschriftmateriaal, verzameld door het Nederlands Forensisch Instituut. De handschriften zijn gemaakt door verdachten in strafzaken. Ze zijn geschreven op ongelinieerd papier. Het formaat van de afbeeldingen is 2480 bij 2244 pixels. De 58 gebruikte pagina's voor de test bevatten in totaal 804 tekstregels en 746 regelovergangen. Het gemiddeld aantal regels per document is 13.9 met een standaarddeviatie van 4.3. Uit de pagina's zijn formulierelementen die geen onderdeel van het handschrift vormen handmatig verwijderd. De afbeeldingen zijn voorbewerkt met het Otsu algoritme (Otsu, 1979) om de achtergrondpixels, die geen onderdeel vormen van de tekst, zo veel mogelijk te verwijderen.

In elk van de 58 handschriften is handmatig aangegeven waar de regels zich bevinden met behulp van regelrepresentaties bestaande uit punten die met lijnstukken met elkaar zijn verbonden. Deze representaties zijn vergeleken met de uitvoer van de drie algoritmes. In geval van een goede regelscheiding zou tussen elk paar regelrepresentaties exact één representatie van een regelovergang moeten liggen. Tijdens de test is geteld hoe vaak de volgende drie fouten werden gemaakt:

- *Overgeslagen regelovergangen*: het aantal regelovergangen waarvoor het te testen algoritme geen representatie genereert. Het testalgoritme kent alle representaties van regelovergangen toe aan de regelovergang waar de meeste van zijn coördinaten binnen vallen. Als hierna aan een regelovergang geen representatie is toegekend wordt dit als één

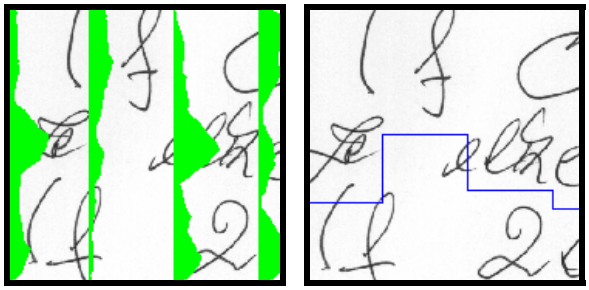


Figuur 3.1: Regelscheiding met Algoritme 1.*

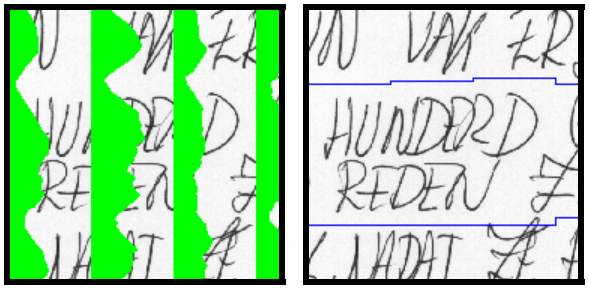
fout gezien.

- *Dubbele regelovergangen*: het aantal regelovergangen waar meer dan één representatie van een regelovergang aan is toegekend.
- *Gesneden regels*: het aantal maal dat een representatie van een regelovergang een regelrepresentatie kruist.

Dezelfde testmethode is gebruikt voor het bepalen van de filterbreedte in algoritme 1 en 2. Hiervoor is een onafhankelijke testset gebruikt, bestaande uit 40 NFI afbeeldingen. Deze set bevatte 602 tekstregels en 562 regelovergangen. Het gemiddeld aantal regels per document bedroeg 15.1 met een standaarddeviatie van 4.1. Er is gekeken voor welke filterbreedte de algoritmes 1 en 2 het minste fouten maken.



Figuur 3.2: Veel ruimte tussen woorden binnen een tekstregel zorgt voor een foutieve gevonden regelovergang.*



Figuur 3.3: Dicht op elkaar geschreven tekstregels zorgen er voor dat de dalen in projectie-histogrammen verdwijnen en de regelovergang niet gevonden wordt.*

3. Resultaten

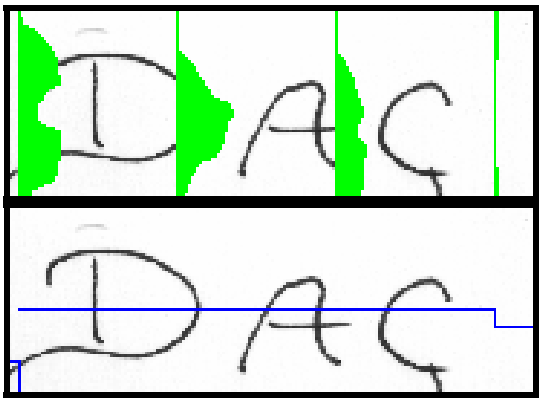
3.1. Visuele impressies

3.1.1. Algoritme 1: Projectiehistogrammen

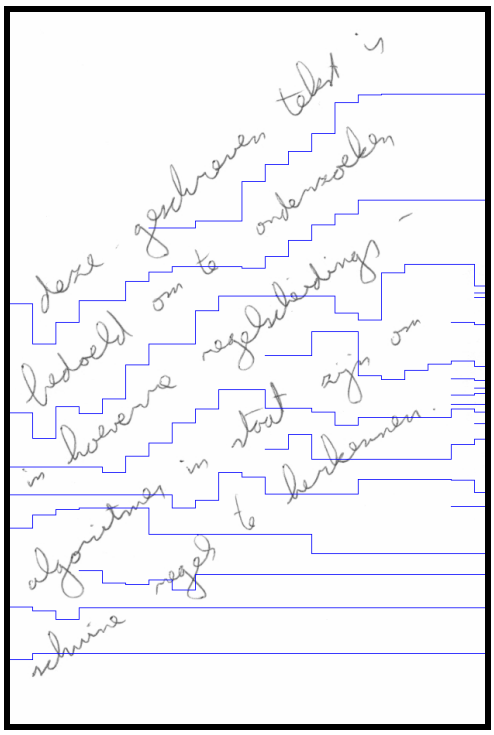
Algoritme 1 detecteert de meeste regel-overgangen correct (figuur 3.1). Het algoritme maakt in een aantal gevallen fouten. Een deel van de fouten wordt veroorzaakt door zinnen waarvan de woorden ver uit elkaar zijn geschreven. Als dit het geval is kan het voorkomen dat een strook tussen de woorden komt te liggen. In dat geval bevat het projectie-histogram op die plaats geen top die de betreffende tekstregel representeert. Er is dus één representatie van een regelovergang te weinig (figuur 3.2).

Daarnaast worden fouten gemaakt als tekst-regels dicht op elkaar zijn geschreven. In dat geval is het dal in het projectiehistogram erg ondiep. Door het filter dat wordt toegepast verdwijnen dalen. Als dat gebeurt wordt de regelovergang niet gevonden (figuur 3.3). Naast vergeten regelovergangen worden soms teveel regelovergangen gevonden. Sommige letters, zoals de 'D' in figuur 3.4 zorgen voor een projectiehistogram met een dal in het midden. Het dal is te breed en te diep om door het filter verwijderd te worden, waardoor het foutief als een regelovergang wordt gezien.

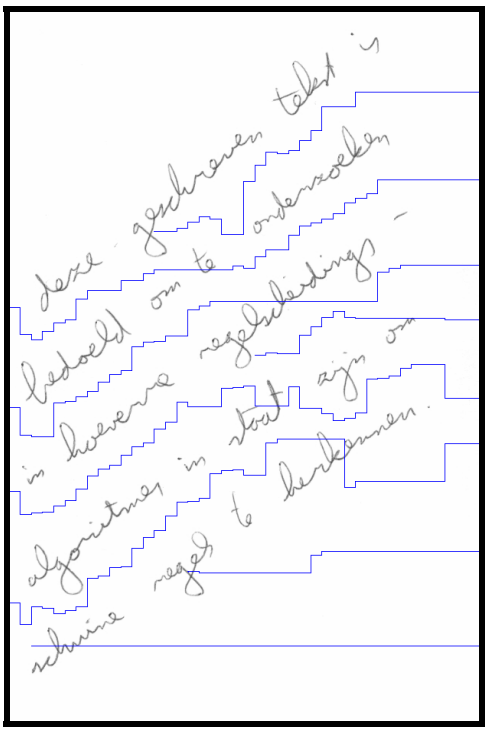
Verder worden veel fouten gemaakt bij documenten met schuine tekstregels (figuur 3.5). De fouten ontstaan door een tweetal redenen. In de eerste plaats zijn de minima ten opzichte van aanliggende stroken verschoven. Er is daarom een verhoogde kans dat een minimum met een verkeerd minimum in de volgende strook wordt



Figuur 3.4: De letter D heeft een projectiehistogram met een dal in het midden. Hierdoor wordt een regelovergang getekend die door de tekst heen snijdt.*

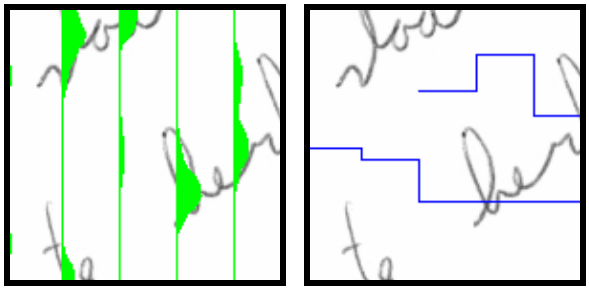


Figuur 3.5: Schuine tekstregels gescheiden door algoritme 1.*

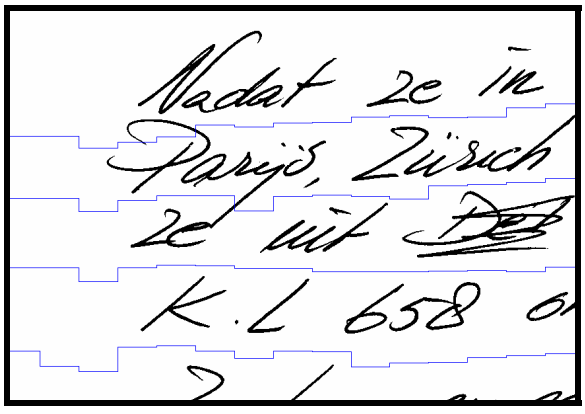


Figuur 3.6: Schuine tekstregels gescheiden door algoritme 2.*

verbonden (figuur 3.7). De tweede reden is dat door schuine regels de toppen die een tekstregel representeren breder worden. Dit betekent dat in het projectiehistogram meer overlap zit tussen de pixels van twee opeenvolgende regels, waardoor de dalen minder diep worden. Er is daarom een grotere kans dat regelovergangen niet gevonden worden.



Figuur 3.7: Schuine tekstregels maken het moeilijker minima correct met elkaar te verbinden.*

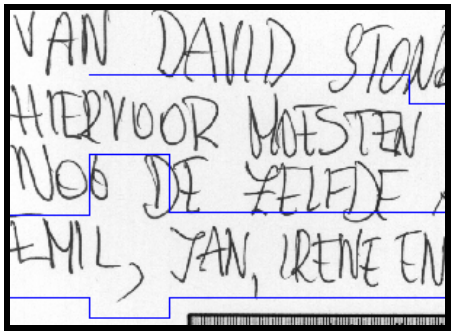


Figuur 3.8: Regelscheiding met Algoritme 2.*

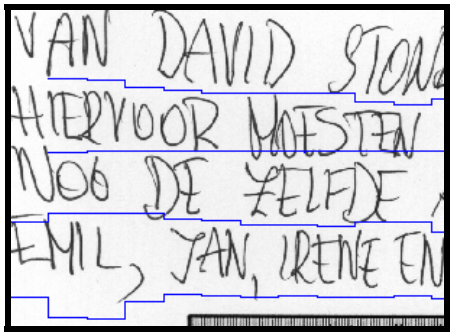
3.1.2. Algoritme 2: Aanpassing Algoritme 1

Algoritme 2 is goed in staat regelovergangen te vinden (figuur 3.8). Daarbij maakt het algoritme veelal dezelfde fouten als algoritme 1. Algoritme 2 is beter in staat regelovergangen te vinden tussen dicht op elkaar geschreven regels. Figuur 3.9 laat een stuk van een tekstdocument zien waarop regelscheiding is toegepast met algoritme 1. De regelovergang tussen de tweede en derde tekstregel is door dit algoritme niet gevonden. Hetzelfde stuk tekst is door algoritme 2 verwerkt in figuur 3.10. Algoritme 2 vindt in dit voorbeeld wel alle aanwezige regelovergangen. Algoritme 2 maakt verder fouten bij schuine tekstregels. Het aantal fouten is echter minder dan bij algoritme 1 (vergelijk figuur 3.5 en 3.6).

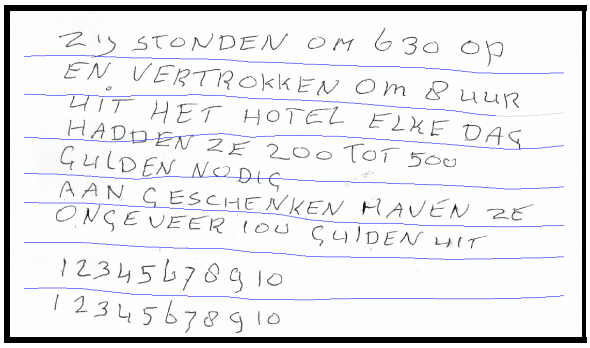
Voor zowel algoritme 1 als voor algoritme 2 geldt dat de breedte van het filter veel invloed heeft op de prestaties. Als een te breed filter wordt gebruikt verdwijnen dalen in het projectiehistogram en worden regelovergangen niet gevonden. Is het filter te smal dan blijven locale minima en maxima in de projectie-histogrammen achter, waardoor teveel regel-overgangen worden gevonden. Voor goede prestaties moet een filter worden gebruikt dat is afgestemd op de schaalgrootte van de tekst. Als de tekst groot geschreven is, moet een breder



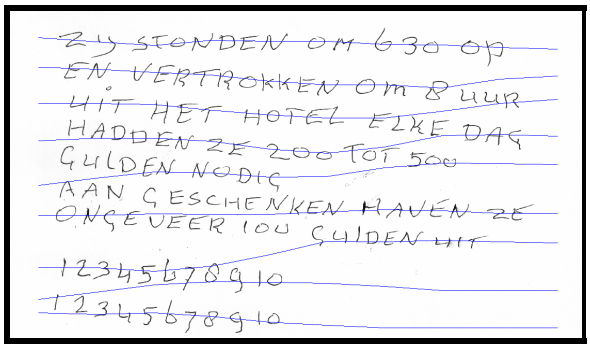
Figuur3.9: Algoritme 1: Eén vergeten regelovergang.*



Figuur 3.10: Algoritme 2: Alle regelovergangen zijn gevonden.*



Figuur 3.11: Regelscheiding met Algoritme 3.*

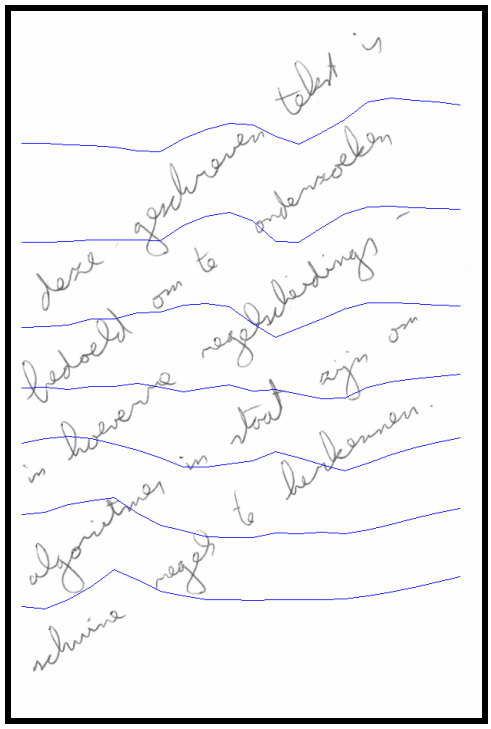


Figuur 3.12 : Regelrepresentaties met Algoritme 4.*

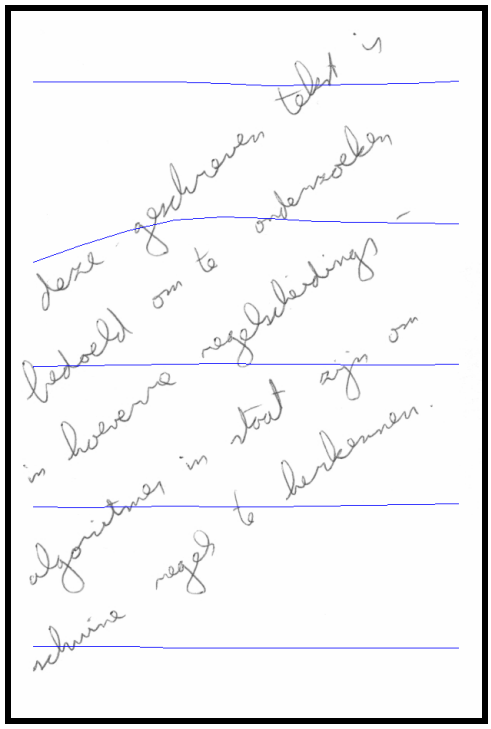
filter worden gebruikt dan wanneer de tekst heel klein geschreven is.

3.1.3. Algoritme 3: KSOM

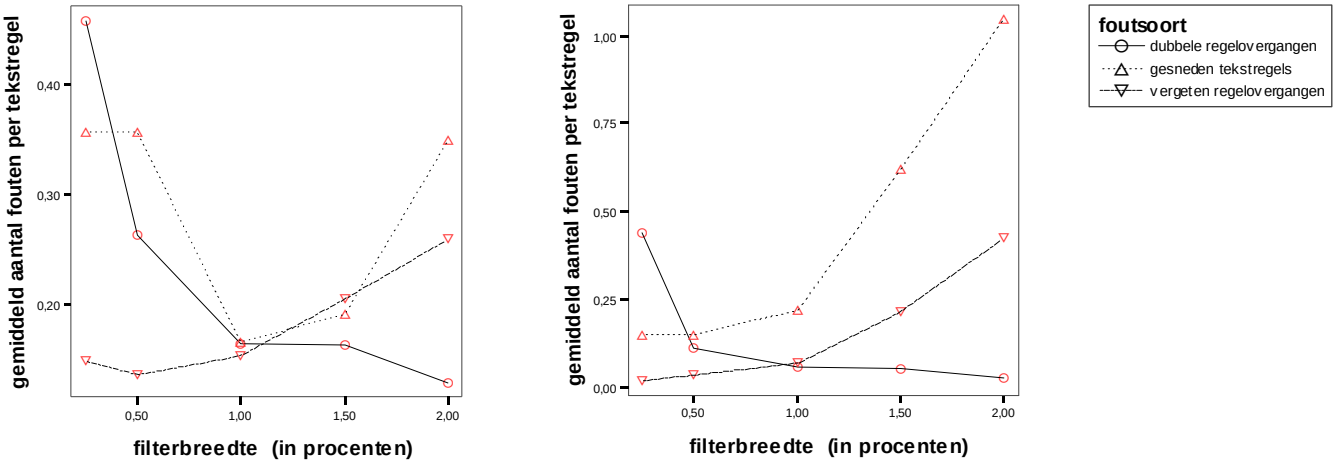
Algoritme 3 is redelijk goed in staat om rechte, horizontale regels van elkaar te scheiden. Figuur 3.11 laat een redelijk goede regelscheiding zien. In dit voorbeeld is één representatie van een regelovergang te weinig gevonden. Dit heeft tot gevolg gehad dat de derde scheidingslijn van boven door een tekstregel heen snijdt. Mogelijk komt dit omdat de regels niet helemaal horizontaal liggen. Het algoritme kan niet omgaan met schuine tekstregels (figuur 3.13). Het netwerk wordt geïnitialiseerd met vrijwel horizontale rijen neuronen. Een aanpassing van een enkel neuron heeft in de eerste tijdstappen tot gevolg dat de neuronen in dezelfde rij een vrijwel even grote aanpassing maken in dezelfde richting als het betreffende neuron. Hierdoor kan



Figuur 3.13: Schuine tekstregels gescheiden door algoritme 3. *



Figuur 3.14: Schuine tekstregels gescheiden met algoritme 4.*



(A) Algoritme 1 **(B) Algoritme 2**

Figuur 3.15: Gemiddeld aantal fouten voor filterbreedtes 0,25 – 2,00 %.

een rij neuronen dat een regel representeert in de eerste tijdstappen niet schuin komen te liggen. In latere tijdstappen kunnen alleen nog kleine aanpassingen worden gemaakt.

3.1.4. Algoritme 4: Genetisch Algoritme

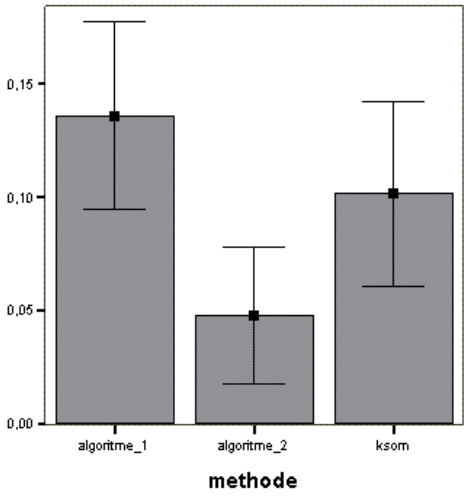
Algoritme 4 werkt vrij slecht. Figuur 3.12 laat in een document zien waar het algoritme de tekstregels lokaliseert. De regelrepresentaties liggen vaak op meer dan één tekstregel. Dit maakt goede regelscheiding onmogelijk. Verder kan algoritme 4 niet omgaan met schuine tekstregels (figuur 3.14). Het algoritme maakt een initiële schatting van het aantal tekstregels. Hierbij wordt soms een fout gemaakt. Het algoritme kan niet compenseren voor teveel of te weinig initieel geschatte tekstregels. Als er te veel of te weinig regelrepresentaties aanwezig zijn heeft dit veelal tot gevolg dat enkele regelrepresentaties op meerdere tekstregels komen te liggen.

De kostenfunctie van het algoritme bestaat uit een lineaire combinatie van drie elementen. Er zijn verschillende combinaties van gewichtswaarden waarmee de afzonderlijke elementen werden vermenigvuldigd uitgeprobeerd. Geen enkele functie kon echter het algoritme goed laten werken. Als er teveel waarde wordt gehecht aan de hoeken tussen de punten in de regelrepresentaties, past het netwerk zich nauwelijks aan. In de initiële toestand zijn de regelrepresentaties relatief recht. Elke aanpassing betekent dat één van de regelrepresentaties minder recht is, waardoor de kosten toenemen. Aan de andere kant zorgt een lage toegekende waarde aan de hoeken er voor dat een regelrepresentatie die op meerdere tekstregels ligt nauwelijks een hogere kostenwaarde krijgt. Als teveel waarde wordt toegekend aan de afstand tussen pixels en de meest nabije regelrepresentatie kan een regelrepresentatie die op een tekstregel is komen te liggen daar moeilijk vandaan komen. Zelfs als een andere regelrepresentatie op dezelfde tekstregel komt te liggen betekent dit nog steeds dat de helft van de pixels binnen die tekstregel het dichtst liggen bij de foutieve regelrepresentatie. Wordt er te weinig waarde aan gegeven dan ontvouwt het netwerk zich niet over het gehele document. Er is geen combinatie gevonden waarmee het algoritme goede regelscheiding kan uitvoeren.

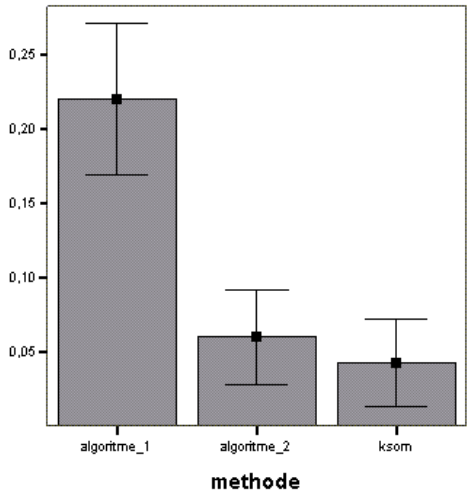
3.2. Kwantificatie

Met behulp van de set van 40 handschriften is bepaald welke filterbreedte het best kan worden gebruikt voor algoritme 1 en 2. Het waargenomen verband tussen filterbreedte en het aantal fouten is weergegeven in figuur 3.15. De breedte is uitgedrukt in procenten van de hoogte van de invoerafbeeldingen. Voor beide algoritmes geldt dat bij een lage filterbreedte er veel dubbele regelovergangen voorkomen. Bij een hoge filterbreedte komen veel vergeten regelovergangen voor. Uiteindelijk is gekozen voor een filterbreedte van 1 % voor algoritme 1 en een filterbreedte van 0.5 % voor algoritme 2.

Na het vaststellen van de parameters zijn de algoritmes 1 tot en met 3 getest met de testset van 58 handschriften. De resultaten zijn

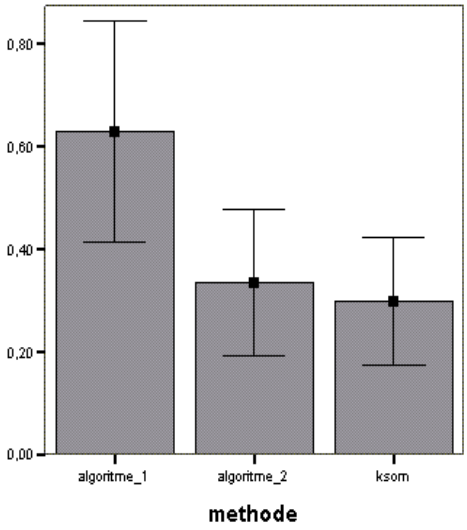


Figuur 3.16: Gemiddeld aantal niet gevonden regelovergangen per daadwerkelijke regelovergang. (95 % bhi's)



Figuur 3.17: Gemiddeld aantal dubbel gevonden regelovergangen per daadwerkelijke regelovergang. (95 % bhi's)

samengevat in de figuren 3.16 tot en met 3.18. Algoritme 2 maakt volgens een paarsgewijze t-toets significant minder *snij-fouten* (gesneden regelrepresentaties) dan algoritme 1 ($t(57) = 3.320$, $p = 0.002$). Ook worden significant minder regelovergangen vergeten ($t(57) = 4.270$, $p < 0.001$) en vaker dan één maal gevonden ($t(57) = 9.429$, $p < 0.001$). Algoritme 3 maakt ook significant minder snij-fouten dan Algoritme 1 ($t(57) = 3.335$, $p = 0.002$). Verder rekent Algoritme 3 significant minder regelovergangen dubbel dan algoritme 1 ($t(57) = 9.715$, $p < 0.001$) en vergeet algoritme 3 minder regelovergangen dan algoritme 1 ($t(57) = 2.513$, $p = 0.015$). Het verschil in snij-fouten tussen algoritme 2 en 3 is niet significant ($t(57) = 0.593$, $p = 0.556$) evenals het verschil in dubbel gevonden regelovergangen ($t(57) = 1.400$, $p = 0.168$). Met algoritme 2 worden wel significant minder regelovergangen niet gevonden dan met algoritme 3 ($t(57) = 3.878$, $p < 0.001$).



Figuur 3.18: Gemiddeld aantal gesneden regels per tekstregel. (95 % bhi's)

4. Discussie en conclusie

Uit de kwantitatieve test is gebleken dat algoritme 2, het aangepaste algoritme op basis van projectiehistogrammen het beste werkt. De visuele impressies van de resultaten ondersteunen dit. Algoritme 2 kan beter omgaan met dicht op elkaar geschreven tekstregels en schuine regels dan algoritme 1, het oorspronkelijke algoritme op basis van projectiehistogrammen. Algoritme 3 op basis van KSOM maakt in vergelijking met algoritme 1, ook minder fouten in de test. Uit visuele impressies blijkt echter dat algoritme 3 minder goed om kan gaan met schuine tekstregels. Daarom kan algoritme 3 niet worden gezien als een betere oplossing dan algoritme 1. Algoritme 4, het genetisch algoritme, vormt geen goede methode om regels te scheiden.

Algoritme 2 maakt significant minder fouten dan algoritme 1. Deze verbetering laat zien dat het relatief eenvoudig is om verbeteringen aan te brengen aan het oorspronkelijke algoritme op basis van projectiehistogrammen. Het is echter de vraag of met dergelijke aanpassingen fundamentele problemen van het algoritme kunnen worden opgelost. De prestaties van de algoritmes 1 en 2 hangen sterk af van de juiste breedte van het gebruikte filter. Voor optimale prestaties zou de filterbreedte moeten afhangen

van de hoogte van de letters in de tekst. Om deze hoogte te bepalen moet bekend zijn waar de tekstregels zich bevinden. Dit is een fundamenteel probleem van dit algoritme, wat suggereert dat er beter kan worden gezocht naar andere manieren om tekstregels te scheiden dan naar manieren om het huidige algoritme te verbeteren.

Aan tekstregels kunnen een aantal kenmerken worden toegeschreven op basis waarvan scheidingsalgoritmes de tekst proberen op te delen. Sommige algoritmes gebruiken voornamelijk een enkel kenmerk, anderen gebruiken meerdere kenmerken. De kenmerken van een tekstregel die in de meeste algoritmes worden gebruikt zijn dat regels relatief veel donkere pixels bevatten (met de aanname dat de tekst met donkere inkt op een lichte achtergrond is geschreven), daarnaast liggen elementen in het verlengde van elkaar, is de afstand tussen regels ongeveer constant en zijn regels bij benadering recht. Deze kenmerken zijn in handgeschreven tekst niet zo sterk aanwezig als in machinetekst, wat goede automatische regelscheiding voor handschrift moeilijk maakt.

Wil een scheidingsalgoritme goed kunnen omgaan met handgeschreven tekst dan moeten verschillende kenmerken van tekstregels worden gebruikt. Alleen dichtheid van donkere pixels is niet voldoende om regelscheiding goed uit te voeren. Zowel algoritme 1 en 2 als algoritme 3 buiten met name dit kenmerk van tekstregels uit. Algoritme 1 en 2 gebruiken daarnaast het kenmerk van continuïteit van tekstregels om de minima in de projectiehistogrammen te verbinden. Zij maken hierbij echter de aanname dat tekstregels bij benadering horizontaal zijn, wat niet altijd het geval is. Daarom kunnen deze algoritmes niet goed omgaan met erg schuine tekstregels. De algoritmes kijken alleen op locaal niveau (tussen twee opeenvolgende stroken) naar (horizontale) continuïteit. De continuïteit van de rest van de regelovergangen of tekstregels wordt niet gebruikt. Als deze kennis zou worden gebruikt zouden fouten als in figuur 3.7 kunnen worden voorkomen.

Ook algoritme 3 op basis van KSOM buit de dichtheid van donkere pixels binnen tekstregels uit. Dit algoritme werkt echter alleen met een erg brede neighborhood om de continuïteit van de regelrepresentaties te waarborgen. Het probleem van deze oplossing is dat het netwerk hierdoor zijn flexibiliteit verliest. Met als gevolg dat het algoritme niet om kan gaan met schuine tekstregels.

Voor het uitvoeren van regelscheiding lijken dus meerdere kenmerken van tekstregels noodzakelijk. Een systeem dat automatisch regelscheiding kan uitvoeren moet dus meerdere kenmerken serieus nemen in plaats van een enkel kenmerk uit te buiten. Om goed te kunnen functioneren is het daarnaast noodzakelijk dat die verschillende elementen op een goede manier in het systeem worden geïntegreerd. Dit is moeilijk gebleken in algoritme 4. In dit algoritme wordt berekend hoe goed kandidaat-oplossingen voldoen aan drie kenmerken van tekstregels en drukt dit uit in een drietal waarden. De resultaten laten zien dat het niet voldoende is om een lineaire combinatie van deze drie waarden te nemen en de uitkomst daarvan te beschouwen als een maat voor de correctheid van de oplossing. Vervolgonderzoek zou meer inzicht kunnen geven over de manier waarop informatie van verschillende kenmerken van tekstregels kan worden geïntegreerd in een enkel algoritme voor regelscheiding.

Referenties

Arivazhagan, A., Srinivasan, H., & Srihari, S. (1997). A statistical approach to line segmentation in handwritten documents. *Document Recognition and Retrieval XIV*, 6500.

Brink, A., Schomaker, L., & Balacu, M. (2007). Towards explainable writer verification and identification using vantage writers. *ICDAR*, 2, 824-828.

Likforman-Sulem, L., Zahour, A., & Taconet, B. (2007). Text line segmentation of historical documents: a survey. *International Journal on Document Analysis and Recognition*, 9, 123-138.

Nicolas, S., Paquet, T., & Heutte, L. (2004). Text line segmentation in handwritten document using a production system. *Frontiers in Handwriting Recognition*, 245-250.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE trans. on Systems, Man and Cybernetics*, 9(1), 62-66.

Shapiro, V., Gluhchev, G., & Sgurev, V. (1993).Handwritten document image segmentation and analysis. *Pattern Recognit. Lett.*, 14, 71-78.

*** Handschrift afkomstig uit de NFI dataset.**