

Overzicht

We geven hier een kort overzicht van de inhoud van deze thesis. De thesis bestaat uit twee grote delen: een literatuurstudie en een uitwerkingsdeel. De delen worden telkens voorafgegaan door een voorblad.

In het eerste deel hebben we kennis gemaakt met allerlei technieken die we kunnen gebruiken om visuele informatie te beschrijven en te vergelijken. Met name wavelets en graaf-matching technieken zijn hierbij interessant.

In het tweede deel hebben we het werken met graaf-algoritmen verder uitgediept en ook extra literatuur verkend omtrent de menselijke visuele perceptie.

Voor meer details en inhoudelijke situering verwijzen we door naar de inleidingen van beide delen.

Dankwoord

Ik wil mijn promotor bedanken voor zijn feedback, want door zijn opmerkingen is de geproduceerde tekst vollediger.

DEEL 1

Academiejaar 2008-2009

Abstract

Een groot deel van de tekst gaat over het fenomeen *wavelets*. Dit is een wiskundige techniek om informatie uit digitale signalen te halen. Een digitaal signaal bestaat uit een eindige opeenvolging van samples. Door het uitvoeren van zogenaamde *wavelet-analyse* is het mogelijk om het verloop van deze samples te beschrijven met een aantal deelsignalen die opgeteld terug het oorspronkelijke signaal geven. Het bijzondere van deze deelsignalen is het feit dat zij bekomen worden door het signaal op verschillende *resolutie-niveaus* te beschouwen: op een laag resolutieniveau wordt een ruw deelsignaal, met zeer weinig details, afgeleid uit het oorspronkelijke signaal dat slechts de globale vorm beschrijft van het oorspronkelijke signaal. Op hogere resolutie-niveaus kunnen zogenaamde verschilsignalen afgeleid worden uit het oorspronkelijke signaal die bij dat eerste ruwe signaal moeten opgeteld worden om terug het oorspronkelijke signaal te bekomen. Het beschouwen van een signaal op meerdere resolutie-niveaus noemt men *multiresolutie*. We bespreken de algemene wiskunde achter *orthonormale wavelets* en ook hoe deze kunnen geïmplementeerd worden met *convolutie*.

In de context van similarity searching op images speelt multiresolutie een bijzondere rol. Mensen lijken in staat om *gelijkaardige* vormen met elkaar te associëren door denkbeeldige abstracties te maken van deze vormen en enkel de meest "opvallende" trekjes van een vorm A te matchen op de meest opvallende trekjes van een vorm B. De meest opvallende trekjes kunnen in het begin bijvoorbeeld de meest ruwe, globale vormen zijn (zoals de armen van een mens). Daarna kan eventueel meer detail in rekening gebracht worden, zoals de vingers van deze mens of een bepaalde grilligheid op een lijn. Nadat we de algemene theorie van wavelets hebben besproken in de eerste hoofdstukken wordt het wavelet-gedeelte van deze thesistekst afgesloten met een toepassing in image searching.

In de laatste drie hoofdstukken van de thesistekst worden ook andere topics binnen similarity searching besproken die helemaal los staan van wavelets. De laatste twee hoofdstukken zijn vooral interessant: zij gaan over het gebruik van grafen om complexe visuele informatie te modelleren. Het is mogelijk om gelijkaardige visuele informatie in twee datasets te ontdekken door een soort van *graph-matching* uit te voeren. Dit lijkt een bijzonder krachtige en flexibele techniek. We denken hier volgend jaar een implementatie over te maken.

Voorwoord

Wavelets zijn een krachtige numerieke techniek en deze tekst biedt natuurlijk slechts een inleiding tot dit onderwerp. Er is veel aandacht besteed aan het begrijpen van wavelets omdat deze een zeer nuttige numerieke tool zijn, die ook buiten deze thesis kan gebruikt worden. Bovendien is het leerrijk geweest om zelf dit domein te leren kennen en op te zoek te gaan naar goede literatuur. Het is daarnaast ook een zeer leerrijke ervaring geweest om al deze wiskundige kennis begrijpbaar proberen neer te schrijven in deze tekst. Wavelets zijn echter een zeer ruim (en ingewikkeld) domein en deze tekst blijft noodzakelijkerwijs aan de oppervlakte. Dit verslag kan eventueel gebruikt worden als een mogelijke introductie tot dit onderwerp.

INHOUDSOPGAVE

ABSTRACT	1
VOORWOORD.....	2
1. OPSTART-WISKUNDE.....	9
1.1. Functies manipuleren	9
1.2. Functieruimte L2	10
2. INLEIDING TOT WAVELETS.....	16
2.1. Overzicht wavelet-categorieën	16
2.2. Gebruik van wavelets	16
2.3. Intuïtieve wavelet-wiskunde	18
3. MULTIRESOLUTIE.....	23
3.1. Maak functies met “Scaling functions”	23
3.2. In de praktijk	30
3.3. Multiresolutie Approximatie van functies	31
3.4. Wavelets	33
3.4.1. Wavelet-analyse	36
3.4.2. De wavelet vergelijking	38
4. CONVOLUTIE	41
4.1. Inleiding tot convolutie	41
4.2. Eigenschappen van convolutie	44
4.3. Snelheidstrucje	45
4.4. Betekenis van convolutie	46
4.5. Tijdscomplexiteit	49
4.6. Downsampling	50
4.7. Upsampling	53

5. DE SNELLE WAVELET TRANSFORMATIE	55
5.1. Analyse.....	55
5.2. Tijdscomplexiteit Analyse-gedeelte	59
5.3. Synthese.....	62
5.4. Implementatie aspecten.....	64
6. NABESPREKING WAVELET TRANSFORMATIE	66
6.1. Fourier transformatie	66
6.2. Convolutie is vermenigvuldiging in frequentie-domein	69
6.3. Link naar filter banks	72
7. IMAGE QUERYING MET WAVELETS	74
7.1. 2D Wavelet Decompositie	75
7.2. Image Querying	88
7.2.1. Image querying afstandsfunctie.....	95
7.2.2. Image querying algoritme	98
8. KORTE INLEIDING TOT SIMILARITY TECHNIEKEN	101
8.1. Onderscheid kwantitatief-kwalitatief maatstaven	101
8.1.1. Voorbeeld kwalitatieve maat.....	101
9. A COMPLEX NETWORK-BASED APPROACH FOR BOUNDARY SHAPE ANALYSIS	105
9.1. Inleiding tot Complexe netwerken	105
9.1.1. Graaf-terminologie.....	106
9.1.2. Metingen en eigenschappen van complexe netwerken	107
9.2. Complexe netwerken om een shape te beschrijven.....	109
9.3. Bespreking.....	114
10. ASSOCIATION GRAPHS.....	119
10.1. Inleiding	119
10.1.1. Exacte methoden.....	120
10.1.2. Niet-exacte methoden.....	124

10.2. Spatial Scene Similarity	124
10.2.1. Semantisch Netwerk en formulering matching probleem	125
10.2.1.1. <i>Concreet matching algoritme</i>	130
10.2.2. Voorbeeld query en oplossing	135
11. MAXIMAL CLIQUE FINDING.....	139
11.1. Basisalgoritme.....	139
11.2. Uitbreidingen	142
11.3. Bespreking implementatie.....	145
12. PERSPECTIEF OP DEEL 2.....	147
12.1. Beschrijving eigen applicatie	147
12.2. Overzicht van mogelijke technieken	148
12.3. Eventuele theoretische uitbreidingen.....	150
12.4. Planning	151
13. BIBLIOGRAFIE.....	152
14. APPENDIX.....	154
14.1. Uitwerking van het BK-algoritme.....	154

Overzicht belangrijke definities

Definitie 1: orthonormale familie.....	12
Definitie 2: orthonormale basis.....	12
Definitie 3: Multiresolutie Approximatie (benadering)	31
Definitie 4: Orthogonaal Complement.....	34
Definitie 5: Downsampling	50
Definitie 6: upsampling.....	53
Definitie 7: Maximal Common Induced Subgraph	121
Definitie 8: maximale, niet-maximale en maximum clique.....	122
Definitie 9: Semantisch Netwerk	126

Overzicht belangrijke stellingen

Stelling 1 : Wavelet Decompositie	58
Stelling 2: Convolutie-stelling	70

Overzicht Animaties

Animatie 1: evolutie van een voorbeeld complex netwerk	111
--	-----

Overzicht Database rankings

Ranking 1: voor de vis-query	117
Ranking 2: voor de spiraal-query	118

Overzicht (pseudo-)codefragmenten

Listing 1: Image Querying zoekalgoritme	99
Listing 2: Maximal Clique Heuristic, version 1	142
Listing 3: Maximal Clique Heuristic, version 2, with pivot selection	145

Overzicht Figuren

Figuur 1: schalen en verschuiven van een moederfunctie	10
Figuur 2: overzicht van de belangrijke hoofdcategorieën van wavelets.....	16
Figuur 3: een discreet signaal voorgesteld als een sequentie van waarden	17
Figuur 4: moeder bultfunctie $a(t)$	19
Figuur 5: moeder bultfunctie $a'(t)$ met ook negatieve waarden.....	19
Figuur 6: signaalfunctie $f(t)$	20
Figuur 7: gebruikte basisvectoren om het signaal in Figuur 6 te beschrijven.....	20
Figuur 8: signaal $f(t)$ getoond samen met zijn ontbinding.....	21
Figuur 9: box-function $b(t)$ getoond als een samenstelling van kleinere box-functions $b(2t)$ en $b(2t-1)$	26
Figuur 10: nesting van de scaling function ruimten	27
Figuur 11: Daubechies Db4 scaling function en wavelet	34

Figuur 12: Daubechies Db2 scaling function en wavelet	34
Figuur 13: de Haar-wavelet.....	40
Figuur 14: Impuls op $t = 0$	46
Figuur 15: datasequentie g (links) en filter f	48
Figuur 16: voor elk sample van de datasequentie g wordt een geschaleerde kopie van de filter gemaakt. Deze zijn hier voor de duidelijkheid onder elkaar weergegeven, maar men moet zich dus inbeelden dat ze allemaal op de grijze tijdslijn rusten. De horizontale positie op de tijd-as geeft aan wanneer de geschaleerde kopie begint in de tijd.	49
Figuur 17: deze figuur toont het resultaat van de convolutie, wanneer alle geschaleerde filterkopieën zijn opgeteld die op dezelfde tijdindex vallen.....	49
Figuur 18: 3 iteraties van het analyse-algoritme. Op het einde neemt de array van berekende approximatie en wavelet coëfficiënten ongeveer dezelfde plaats in als input approximatie-coëfficiënten.....	60
Figuur 19: signal extension technieken; (a) zero-padding, (b) periodic, (c) symmetric.....	65
Figuur 20: positieve-frequentie fasor.....	68
Figuur 21: projectie levert gewone sinus en cosinus op.....	68
Figuur 22: perfect reconstruction filter bank structuur	73
Figuur 23: verband tussen de perfect reconstruction filters	73
Figuur 24: highpass en lowpass filter in frequentie domein.....	73
Figuur 25: coëfficiënten na twee iteraties van de één-dimensionale decompositie op één rij uit een grid van samples.	76
Figuur 26: coëfficiënten na twee iteraties van de één-dimensionale decompositie op één kolom uit een grid van samples.	76
Figuur 27: Stap 1 van de 2D standaard-decompositie. De verdeling van de coëfficiënten is getoond na twee iteraties van de één-dimensionale decompositie op elke rij uit een grid van samples.	76
Figuur 28: Stap 2 van de 2D standaard-decompositie. De verdeling van de coëfficiënten is getoond na twee iteraties van de één-dimensionale decompositie op elke kolom.	77
Figuur 29: 2D standaard basis voor $V_j \otimes V_j$, gemaakt uit de Box scaling function en bijhorende Haar wavelet. Deze figuur is gebaseerd op (Stollnitz, Derose, & David, 1996).	81
Figuur 30: opdeling in zones na twee iteraties van de 2D standaard decompositie. Per zone is de 2D kern-basisvector gevisualiseerd.	83
Figuur 31: zone 6 uit Figuur 30.....	83
Figuur 32: stap 1 in iteratie 1 van de 2D niet-standaard decompositie.....	84
Figuur 33: stap 2 in iteratie 1 van de 2D niet-standaard decompositie.....	84
Figuur 34: stap 1 in iteratie 2 van de 2D niet-standaard decompositie.....	84
Figuur 35: stap 2 in iteratie 2 van de 2D niet-standaard decompositie.....	84
Figuur 36: de formules van de 2D basisvectoren gevoegd bij de zones uit Figuur 35.	85
Figuur 37: opdeling in zones na twee iteraties van de 2D niet-standaard decompositie. Per zone is de 2D kern-basisvector gevisualiseerd.	85
Figuur 38: testfiguur, een standaard beeldverwerkingfoto van een vrouw met een kap.....	87
Figuur 39: 2D niet-standaard wavelet decompositie (2 iteraties) van Figuur 38 met de Box scaling function en Haar wavelet.	87
Figuur 40: testfiguur, een rechthoek met afgeronde hoeken. Aan de afgeronde hoeken ontstaan een soort van diagonale randen die we kunnen detecteren met een 2D wavelet decompositie. ...	88
Figuur 41: 2D niet-standaard decompositie (5 iteraties) van Figuur 40 met de <i>Coiflet</i> scaling function en wavelet (type 1).....	88
Figuur 42: het omzetten van elke afbeelding naar een gemeenschappelijke vierkante grootte. ...	90
Figuur 43: volledige 2D niet-standaard decompositie van Figuur 38 met de Box scaling function en Haar wavelet.	94
Figuur 44: truncation toegepast op de coëfficiënten in Figuur 43.....	94
Figuur 45: deze afbeelding is het resultaat na het toepassen van een 2D wavelet-synthese op de geselecteerde coëfficiënten in Figuur 44.....	95
Figuur 46: ruwe topologische relaties tussen twee veelhoeken zonder gaten.	102
Figuur 47: conceptual neighborhood graph van high-level topologische relaties	103

Figuur 48: complex netwerk naar feature vector.....	107
Figuur 49: evolutie van een complex netwerk. Merk op dat er ook geïsoleerde knopen mogen bijkomen (en verdwijnen)	108
Figuur 50: een verzameling punten in het vlak, in de vorm van een ster. Dit voorbeeld is gebaseerd op (Ricardo Backes, 2009).....	111
Figuur 51: query-patroon, de vorm van een vis	116
Figuur 52: query patroon, een spiraalachtige lijn	116
Figuur 53: twee voorbeeld-grafen G en G'	123
Figuur 54: de twee maximale common induced subgraphs voor het grafen-paar van Figuur 53.	123
Figuur 55: dit is geen maximale common induced subgraph voor het grafen-paar van Figuur 53.	123
Figuur 56: voorbeeld-configuratie	127
Figuur 57: voorbeeld semantisch netwerk dat hoort bij de objecten van Figuur 56.	128
Figuur 58: high-level topologische relaties geschreven op de bogen vanuit knoop A in het semantisch netwerk van Figuur 57.	128
Figuur 59: een database-configuratie van objecten die de gebieden voorstellen op een campus. Blauw: parkeerplaatsen van academisch personeel. Rood: residentiële parking. Grijs: academische gebouwen. De andere kleuren zijn niet belangrijk.	136
Figuur 60: een query-configuratie. Er is een academisch gebouw X dat raakt aan een personeelsparking, die zelf ook weer raakt aan een residentiële parking. Het academische gebouw raakt de residentiële parking niet.	136
Figuur 61: association graph	137
Figuur 62: de gematchte objecten uit de database-configuratie. Dit is een <i>volledige</i> oplossing voor de query in Figuur 60.	138
Figuur 63: voorbeeld-graaf.....	139
Figuur 64: voorbeeld-graaf.....	154

1. Opstart-Wiskunde

In dit hoofdstuk zullen we kort enkele wiskundige begrippen introduceren waar we in latere hoofdstukken van zullen gebruik maken.

1.1. Functies manipuleren

Stel, we hebben een functie $f : \mathbb{R} \rightarrow \mathbb{R} : t \mapsto f(t)$ waarbij t de veranderlijke is. We kunnen deze functie horizontaal verschuiven volgens de t -as met stapgrootte b naar rechts (of $-b$ naar links):

$$f(t-b) \quad b \in \mathbb{Z} \quad (1.1)$$

Ook kunnen we haar horizontaal schaleren volgens de t -as:

$$f(at) \quad a \in \mathbb{Q} \quad (1.2)$$

Wanneer $a > 1$, zal $f(at)$ smaller zijn dan $f(t)$.

Wanneer $a < 1$, zal $f(at)$ breder zijn dan $f(t)$.

En bovenstaande "manipulaties" kunnen we ook tegelijk doen:

$$f(at-b) \quad a \in \mathbb{Q}, b \in \mathbb{Z} \quad (1.3)$$

Dit laatste kunnen we bekijken als:

- eerst een horizontale schaling met factor a , gevolgd door een horizontale verschuiving met stapgrootte b/a naar rechts ofwel
- als een horizontale verschuiving met stapgrootte b , gevolgd door een horizontale schaling met factor a .

Als b constant gehouden zou worden, dan geldt: hoe groter a , hoe kleiner de translatie ten gevolge van b zal zijn.

We kunnen $f(t)$ beschouwen als een "moeder"-functie waar we oneindig veel kindfuncties mee kunnen creëren door $f(t)$ te verschuiven en te schaleren.

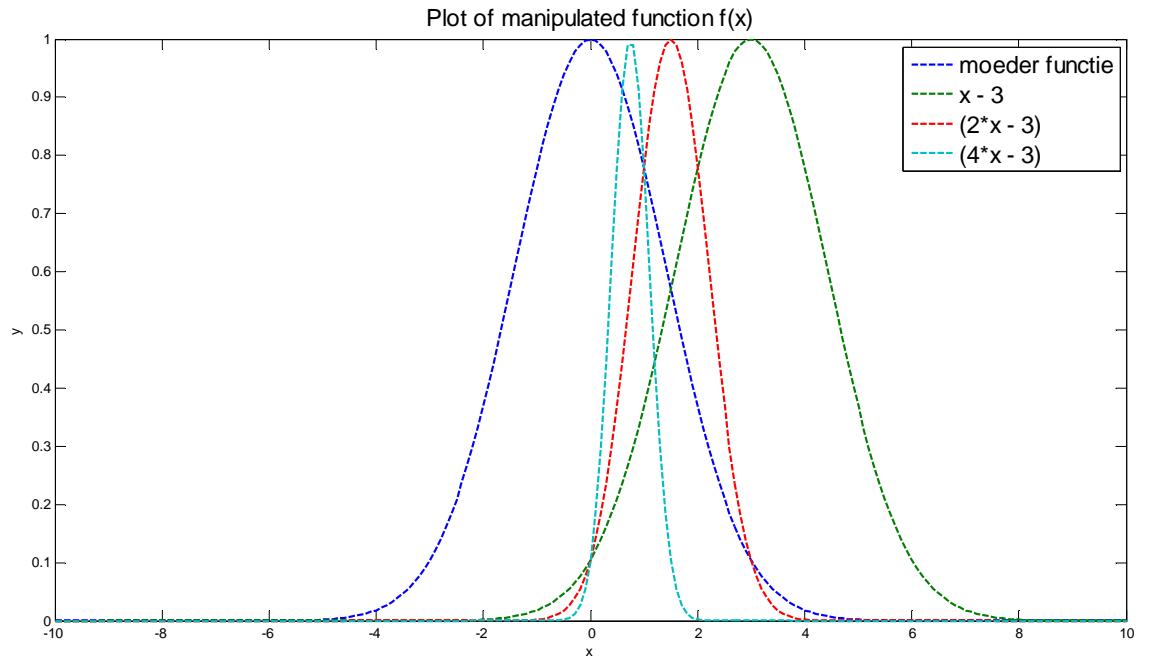
In Figuur 1 wordt dit geïllustreerd. Daar werd een willekeurige moederfunctie genomen en deze werd horizontaal geschaleerd en verschoven volgens bovenstaande formules.

Een deelverzameling $A \subset \mathbb{R}$ wordt *compact* genoemd wanneer zij *gesloten* en *begrensd* is. *Gesloten* (in \mathbb{R}) wil zeggen dat de randpunten ook tot A behoren. *Begrensd* wil zeggen dat A bevat is in een interval van de vorm $[u, v]$, met $u, v \in \mathbb{R}$ en $u < v$.

De plaats op de t -as waar een functie $f(t)$ een waarde verschillend van 0 aanneemt noemt men de "support" van die functie. De support van een functie is dus een deelverzameling van zijn domein. Wanneer deze support compact is, zoals een interval bijvoorbeeld, dan spreken we van "compact support".

Als de moederfunctie compacte support heeft en we deze horizontaal samendrukken, dan heeft de

resulterende kindfunctie een kleinere support dan de moederfunctie. Als we de moederfunctie horizontaal uitrekken heeft de resulterende kindfunctie een grotere support als de moederfunctie.



Figuur 1: schaleren en verschuiven van een moederfunctie

1.2. Functieruimte L2

Deze paragraaf vormt een korte introductie tot de gebruikte wiskundige terminologie. Een boek over de aangeraakte onderwerpen is (Rudin, 1986). Het werk (Koornwinder, 1993) bevat ook een apart hoofdstuk met *mathematical preliminaries*.

Er wordt in een reële vectorruimte H een afstand gedefinieerd tussen twee vectoren met behulp van een *norm*, genoteerd met $\|\cdot\|$. Een normfunctie $\|\cdot\|: H \rightarrow \mathbb{R}^+$ moet aan de volgende voorwaarden voldoen:

$$\begin{aligned} \forall f \in H : \\ \|f\| &\geq 0, \\ \|f\| = 0 &\Rightarrow f = 0 \end{aligned} \tag{1.4}$$

$$\begin{aligned} \forall f \in H, \forall \lambda \in \mathbb{R} : \\ \|\lambda f\| &= |\lambda| \|f\| \end{aligned} \tag{1.5}$$

$$\begin{aligned} \forall f, g \in H : \\ \|f + g\| &\leq \|f\| + \|g\| \end{aligned} \tag{1.6}$$

Een ruimte die een dergelijke norm definiert noemt men een *Banach*-ruimte. De voorwaarde is wel dat de norm *volledig* is. Dit wil zeggen dat de limiet van een oneindige rij van vectoren (uit die vectorruimte) bestaat wanneer de afstand tussen twee opeenvolgende vectoren in die rij steeds kleiner wordt. Zulke rij wordt een Cauchy rij genoemd.

Maar we willen buiten een norm ook graag een *inner product*, genoteerd $\langle \cdot, \cdot \rangle$, waarmee we kunnen nagaan hoe de vectoren in de vectorruimte staan ten opzichte van elkaar. De Nederlandstalige term voor inner product is *scalair product* of *inproduct*. We spreken dan van hoeken tussen de vectoren en orthogonaliteit. Een *Hilbert*-ruimte is een Banach-ruimte waar ook nog een inner product op gedefinieerd wordt.

Zij H een Banach-ruimte. Het inner product is een functie $\langle \cdot, \cdot \rangle : H \times H \rightarrow \mathbb{R}$, zodat:

$$\begin{aligned} \forall f_1, f_2, g \in H, \forall \lambda_1, \lambda_2 \in \mathbb{C}: \\ \langle \lambda_1 f_1 + \lambda_2 f_2, g \rangle = \lambda_1 \langle f_1, g \rangle + \lambda_2 \langle f_2, g \rangle \end{aligned} \quad (1.7)$$

$$\begin{aligned} \forall f \in H: \\ \langle f, f \rangle \geq 0 \wedge \\ \langle f, f \rangle = 0 \Leftrightarrow f = 0 \end{aligned} \quad (1.8)$$

$$\forall f, g \in H: \langle f, g \rangle = \langle g, f \rangle \quad (1.9)$$

Men kan uit een inner product een norm definiëren:

$$\|f\| = \langle f, f \rangle^{1/2} \quad (1.10)$$

De verzameling van alle kwadraat-integreerbare functies wordt genoteerd met $L^2(\mathbb{R})$ of zelfs kortweg L2. Dit wil zeggen dat de Lebesgue-integraal van het kwadraat van de absolute waarde van een dergelijke functie over een bepaald interval eindig is. De vorm van de functies is $f : \mathbb{R} \rightarrow \mathbb{R}$.

Omdat we lineaire combinaties van dergelijke functies kunnen nemen die terug in $L^2(\mathbb{R})$ zitten, is $L^2(\mathbb{R})$ per definitie een vectorruimte. De elementen van $L^2(\mathbb{R})$ kunnen we gewoon functies noemen, maar de term vector is ook toegelaten. $L^2(\mathbb{R})$ is een Hilbert-ruimte. Het inner product wordt als volgt gedefinieerd ($f, g \in L^2(\mathbb{R})$):

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(t)g(t)dt$$

En de norm is dus:

$$\|f\|^2 = \int_{-\infty}^{\infty} |f(t)|^2 dt \Rightarrow \|f\| = \left(\int_{-\infty}^{\infty} |f(t)|^2 dt \right)^{1/2} \quad (1.11)$$

We zullen in deze tekst bijna uitsluitend $t \in \mathbb{R}$ als de veranderlijke gebruiken voor de functies. Door de norm te nemen van het verschil van twee functies kunnen we uitdrukken wanneer twee functies f_1 en f_2 in $L^2(\mathbb{R})$ aan elkaar gelijk zijn. Inderdaad, we hebben

$$\|f_1 - f_2\| = \left(\int_{-\infty}^{\infty} |f_1(t) - f_2(t)|^2 dt \right)^{1/2} = 0$$

Met $f_1 = f_2$ bedoelen we $f_1(t) = f_2(t)$ voor elke $t \in \mathbb{R}$. Er geldt dat $\|f_1 - f_2\| = 0 \Leftrightarrow f_1 = f_2$. Dit geldt trouwens voor elke vectorruimte waarop we een norm gedefinieerd hebben.

Definitie 1: orthonormale familie

Een oneindige familie van functies $\{e_n\}_{n \in \mathbb{N}}$ uit $L^2(\mathbb{R})$ noemt men *orthonormaal* wanneer

$$\forall a, b \in \mathbb{N} : a \neq b \Rightarrow \langle e_a, e_b \rangle = 0$$

$$\forall n \in \mathbb{N} : \|e_n\| = 1$$

Dus het inner product van elk paar verschillende leden moet nul zijn en de norm van elk lid is gelijk aan één. Wanneer we de laatste voorwaarde laten vallen, dan spreken we van een *orthogonale* familie.

Merk op dat we met de notatie $\{e_n\}_{n \in \mathbb{N}}$ niets zeggen over de specifieke vorm van de functies e_n .

Bovendien wil $n \in \mathbb{N}$ gewoon zeggen dat we een denkbeeldig label (nummertje) toekennen aan elke basisfunctie. De stijgende volgorde van de getallen uit \mathbb{N} zegt niets over de volgorde van de functies e_n op de tijd-as.

Een basis van $L^2(\mathbb{R})$ is een deelverzameling van $L^2(\mathbb{R})$ zodat men door lineaire combinaties van functies uit die deelverzameling te nemen alle functies in $L^2(\mathbb{R})$ kan construeren.

Definitie 2: orthonormale basis

Zij $\{e_n\}_{n \in \mathbb{N}}$ een orthonormale familie van functies uit $L^2(\mathbb{R})$. Indien

$\forall f \in L^2(\mathbb{R})$ er een sequentie $\{\lambda_n\}_{n \in \mathbb{N}}$ van coëfficiënten bestaat zodat

$$\lim_{N \rightarrow \infty} \left\| f - \sum_{n=0}^N \lambda_n e_n \right\| = 0,$$

dan noemt men $\{e_n\}_{n \in \mathbb{N}}$ een *orthonormale basis* van $L^2(\mathbb{R})$. Op een gelijkaardige manier kan men ook een *orthogonale basis* definiëren,

maar we beschouwen verder in de tekst vooral orthonormale basissen.

Deze laatste definitie drukt uit dat we elke coëfficiënt λ_n vermenigvuldigen met een bepaald lid e_n uit de orthonormale familie $\{e_n\}_{n \in \mathbb{N}}$. De term $\lambda_n e_n$ is daardoor een verticaal geschaleerde versie van functie e_n . Wanneer we, gegeven een aantal te gebruiken coëfficiënten N , via de sommatie al deze $\lambda_n e_n$ optellen creëren we een tweede functie, namelijk $g = \sum_{n=0}^N \lambda_n e_n$. We berekenen in de limiet de norm van het verschil van de functies f en g . We drukken met $N \rightarrow \infty$ uit dat door meer coëfficiënten te gebruiken (en dus meer bijhorende basisfuncties) de fout tussen de geconstrueerde functie g en f naar nul zal gaan.

We zien in deze definitie meteen dat deze orthonormale basis $\{e_n\}_{n \in \mathbb{N}}$ oneindig veel functies uit $L^2(\mathbb{R})$ bevat, in tegenstelling tot de orthonormale basissen voor het vlak \mathbb{R}^2 waar precies twee vectoren in elke orthonormale basis zitten.

Als $\{e_n\}_{n \in \mathbb{N}}$ een orthonormale basis is, bestaat voor elke f een sequentie $\{\lambda_n\}_{n \in \mathbb{N}}$ zoals beschreven in definitie 2. De lezer vraagt zich misschien het volgende af:

- Hoe vinden we deze sequentie van $\{\lambda_n\}_{n \in \mathbb{N}}$?
- Zijn er in het algemeen meerdere mogelijke dergelijke sequenties?

We gaan nu aantonen dat er precies één sequentie $\{\lambda_n\}_{n \in \mathbb{N}}$ bestaat, waarbij deze coëfficiënten als volgt berekend worden:

$$\lambda_n = \langle f, e_n \rangle$$

De eigenschap die hieruit volgt is dat we een functie f mogen schrijven als een lineaire combinatie van de basisvectoren e_n :

$$\begin{aligned} f &= \sum_{n=0}^{\infty} \langle f, e_n \rangle e_n \\ &= \sum_{n=0}^{\infty} \lambda_n e_n \end{aligned} \tag{1.12}$$

Bewijs

Gegeven: zij $\{e_n\}_{n \in \mathbb{N}}$ een orthonormale basis. Zij $f \in L^2(\mathbb{R})$ en $\{\lambda_n\}_{n \in \mathbb{N}}$ een rij getallen zodat aan definitie 2 voldaan is:

$$\lim_{N \rightarrow \infty} \left\| f - \sum_{n=0}^N \lambda_n e_n \right\| = 0$$

We zullen bewijzen dat noodzakelijk geldt: $\lambda_n = \langle f, e_n \rangle$

Dit heeft als gevolg dat er dus slechts één unieke sequentie $\{\lambda_n\}_{n \in \mathbb{N}}$ bestaat.

Bewijs: Definieer de rij vectoren

$$\delta_N := f - \sum_{n=0}^N \lambda_n e_n$$

Het gegeven vertelt ons dat $\lim_{N \rightarrow \infty} \|\delta_N\| = 0$. Voor elke $m < N$ hebben we

$$\begin{aligned} \langle \delta_N, e_m \rangle &= \left\langle f - \sum_{n=0}^N \lambda_n e_n, e_m \right\rangle \\ &= \langle f, e_m \rangle - \sum_{n=0}^N \lambda_n \langle e_n, e_m \rangle \\ &= \langle f, e_m \rangle - \lambda_m \end{aligned}$$

Deze laatste stap geldt wegens de orthonormaliteit van $\{e_n\}_{n \in \mathbb{N}}$: $\langle e_m, e_m \rangle = 1$ en $\forall n \neq m \quad \langle e_n, e_m \rangle = 0$.

Wegens de ongelijkheid van Schwarz geldt:

$$|\langle \delta_N, e_m \rangle| \leq \|\delta_N\| \cdot \|e_m\|$$

Omdat we al weten dat $\lim_{N \rightarrow \infty} \|\delta_N\| = 0$ en $\|e_m\|$ gelijk is aan één, gaat het linkerlid in bovenstaande vergelijking ook naar nul: $\lim_{N \rightarrow \infty} |\langle \delta_N, e_m \rangle| = 0$. Gecombineerd met de eerdere afleiding geldt nu:

$$\lim_{N \rightarrow \infty} |\langle \delta_N, e_m \rangle| = \lim_{N \rightarrow \infty} |\langle f, e_m \rangle - \lambda_m| = 0$$

Maar omdat de grootheid $\langle f, e_m \rangle - \lambda_m$ onafhankelijk is van N mogen we de limiet weglaten:

$$|\langle f, e_m \rangle - \lambda_m| = 0 \Rightarrow \langle f, e_m \rangle = \lambda_m$$

Wegens overzichtelijkheid hebben we in bovenstaande afleiding een m gebruikt om een andere letter te gebruiken dan de n . Maar er geldt dus in het algemeen dat

$$\lambda_n = \langle f, e_n \rangle$$

Q.E.D

In een orthogonale basis worden de λ_n op een iets andere manier berekend:

$$\lambda_n = \frac{\langle f, e_n \rangle}{\|e_n\|^2}$$

Een orthogonale basis kan steeds omgezet worden naar een orthonormale basis door elke basisvector te *normeren* (delen door zijn norm).

Een Hilbert-ruimte waarvoor een orthogonale basis bestaat noemt men *separable* (verdeelbaar). Dit wil zeggen dat we eender welke functie in de Hilbert-ruimte kunnen opsplitsen in verticaal geschaleerde basisvectoren door de bijhorende coëfficiënten te berekenen.

Wanneer we het inner product nemen van een functie $g \in L^2(\mathbb{R})$ met beide kanten van (1.12) en de eigenschappen (1.7) en (1.9) van inner product gebruiken, krijgen we:

$$\begin{aligned} \langle f, g \rangle &= \left\langle \sum_{n=0}^{\infty} \langle f, e_n \rangle e_n, g \right\rangle \\ &= \sum_{n=0}^{\infty} \langle f, e_n \rangle \langle e_n, g \rangle \\ &= \sum_{n=0}^{\infty} \langle f, e_n \rangle \langle g, e_n \rangle \end{aligned}$$

De bekomen vergelijking noemt men de *Parseval gelijkheid* voor orthonormale basissen. Wanneer $f = g$ leiden we hier een interessant feit uit af:

Plancherel formule
$\langle f, f \rangle = \ f\ ^2 = \sum_{n=0}^{\infty} \langle f, e_n \rangle ^2 = \sum_{n=0}^{\infty} \lambda_n ^2$

Met $\|f\|^2$ wordt de zogenaamde "energie" van een functie f uitgedrukt. De Plancherel formule vertelt ons dat in de orthonormale basis de energie van de functie letterlijk bevat zit in de coëfficiënten die we bekomen wanneer we f ontbinden in die basis.

De wavelets die we zo dadelijk zullen tegenkomen vormen onder andere een orthonormale basis van $L^2(\mathbb{R})$, waardoor $L^2(\mathbb{R})$ dus separable is. Wavelet-analyse zal voor een signaalfunctie f de coëfficiënten berekenen in de orthonormale basis. De gebruiker van deze ontbinding kan deze coëfficiënten dan manipuleren door er een paar weg te laten of door sommige coëfficiënten te vergroten of te verkleinen waardoor de invloed van een bepaalde basisvector verhoogd of verlaagd wordt. Daardoor kan men dan een signaalfunctie f' synthetiseren. De Plancherel formule is in dat geval erg nuttig omdat het vrij goed toelaat om het verschil $\|f - f'\|$ te voorspellen aan de hand van het verschil tussen de oorspronkelijke coëfficiënten en de aangepaste coëfficiënten.

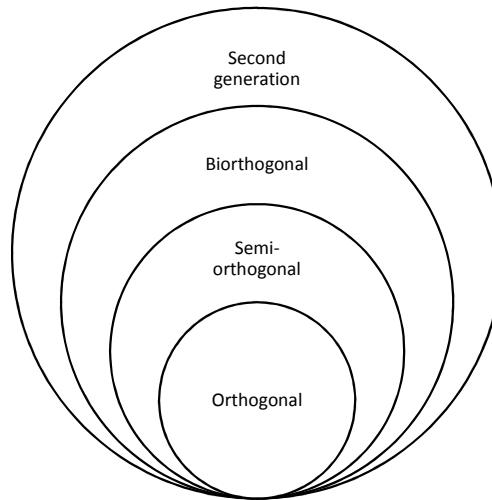
2. Inleiding tot wavelets

Dit hoofdstuk geeft eerst een overzicht van de belangrijkste bestaande wavelet-categorieën en verdergaand op sommige wiskundige begrippen uit vorig hoofdstuk zal aangegeven worden wat de essentie is van wavelet-analyse.

2.1. Overzicht wavelet-categorieën

Het woord "wavelet" betekent vrij vertaald "golfje". De allereerste ideeën over wavelets komen oorspronkelijk uit Frankrijk en daar sprak men van "Ondelette".

In Figuur 2 wordt een overzicht gegeven van een aantal belangrijke hoofdcategorieën van wavelets in de literatuur. Met de nesting van de cirkels worden de veralgemeningen aangegeven. Er bestaan vele soorten wavelets en uitbreidingen binnen elke hoofdcategorie.



Figuur 2: overzicht van de belangrijke hoofdcategorieën van wavelets

In deze tekst zullen we "eerste generatie" orthogonale waveletsystemen bekijken, de binnenste cirkel op de figuur. Deze wavelets behoren tot de eerste wavelets die zijn ontwikkeld en onderzocht. We zullen concreet in deze tekst enkele wiskunde- en implementatie-aspecten bespreken van de *orthonormale* wavelets. Dit zijn genormeerde orthogonale wavelets. In het bijzonder zullen we zelfs enkel orthonormale wavelets met *compacte support* bespreken.

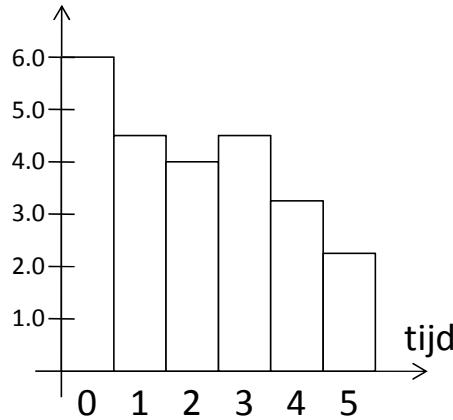
2.2. Gebruik van wavelets

De data waar we wavelet analyse op gaan toepassen is opgeslagen op onze computer in de vorm van een reeks getallen (discrete data). Deze reeks wordt ook wel "signaal" of "databeeld" genoemd. Onze reeks getallen kan bijvoorbeeld simpelweg een rij pixelwaarden van een kleurenkanaal voorstellen van een afbeelding maar we kunnen natuurlijk ook een natuurkundig fenomeen zoals temperatuur samplen over de tijd.

Elke reeks van dergelijke getallen kunnen we plotten in een grafiek. Op de x-as zetten we de indices van de rij uit (meestal natuurlijke getallen). Op de y-as worden de feitelijke getalwaarden uit de sequentie geplaatst, één voor elke index op de x-as. Intuïtief gezien verschijnt er dus een rij paaltjes van verschillende hoogte naast elkaar, zoals getoond in Figuur 3.

Omdat de volgorde van de samples zo belangrijk is, spreken we vaak van een "tijdsordening". De x-as wordt hierna vaak de "tijd-as" genoemd.

In de tekst hieronder worden in de theorie wel vaak continue functies gebruikt voor eenvoudigere notatie. Hieruit leiden we wel telkens af wat er gebeurt in het discrete geval.



Figuur 3: een discreet signaal voorgesteld als een sequentie van waarden

Wavelet analyse komt overeen met een mechanisme om een dasequentie op te delen in verscheidene onafhankelijke deel-sequenties. Het is de bedoeling om de oorspronkelijke sequentie zodanig op te splitsen dat men bepaalde kenmerkende trekjes ervan kan aantreffen in deze deelsequenties. Er is dus een soort van "ontwarrings"-mechanisme nodig. De meeste interessante signalen zijn niet periodiek van aard. Dit wil zeggen dat elke volgende vorm van waarden in het signaal in de tijd weer uniek zal zijn en waarschijnlijk nooit meer precies hetzelfde zal terugkomen. We kunnen dus van een signaal geen model maken en op die manier voorspellen hoe alle toekomstige pieken en andere kenmerken er zullen gaan uitzien naarmate de tijd vordert. Als we toch willen weten wanneer de kenmerken van een signaal voorkomen en deze willen afzonderen en bestuderen, zal ons ontwarrings-mechanisme dus *tijdelijke* fenomenen moeten kunnen onderscheiden, door enkel gebruik te maken van de waargenomen tijdelijke informatie.

Maar we voelen al aan dat er verscheidene schaal-niveaus zijn waarop we de wavelet analyse zullen moeten uitvoeren: om de grote en trage bewegingen van het signaal op te merken zullen we op een andere schaal naar het signaal moeten kijken dan wanneer we naar de meer frequente bewegingen zoals kleine "piekjes" in de waarden willen kijken. Het complex van vormen in het signaal kunnen we beschrijven als een fenomeen dat bestaat uit een aantal "lagen" van detail. Op het meest ruwe niveau kunnen we een signaal beschrijven als traag variërende vormen in de waarden. Op meer gedetailleerde niveaus voegen we dan ook kleinere piekjes in de waarden toe. Deze laatsten zijn meestal "noise" die op ons signaal zit en waar we soms vanaf willen geraken. In ideale omstandigheden zullen we soms een vervuild signaal kunnen zuiveren door de traag variërende kenmerken over te houden en de zeer hoog frequente noise af te scheiden. Het is wenselijk dat afgescheiden noise zogenaamde "white noise" is, waar geen vorm van een signaal meer in te herkennen valt. Onwenselijk is meestal het geval wanneer er wel een soort van signaal te herkennen zou zijn in de afgescheiden noise, wat we dan "coloured noise" noemen.

2.3. Intuïtieve wavelet-wiskunde

In deze paragraaf proberen we enkel het basisidee van wavelet-analyse te introduceren. De verwoording en de figuren zijn zeer eenvoudig gehouden en hebben daarom geen wiskundig exacte juistheid.

Stel, we hebben een continue signaalfunctie $f(t) \in L^2(\mathbb{R})$. Deze functie beschrijft een signaal dat varieert over de tijd. Deze tijd hoeft geen echte klok-tijd te zijn, het kan evengoed een andere variabele aanduiden zoals bijvoorbeeld: t is de afstand van een sonde gereisd door de ruimte en $f(t)$ de overblijvende hoeveelheid brandstof. Voortaan zullen we gewoon spreken over tijd, alle mogelijke interpretaties van t abstraherend. Het domein van de tijd t mag hier nog van $-\infty$ tot $+\infty$ gaan. Voor elke waarde van t zal $f(t)$ de waarde aangeven die het signaal aanneemt op dat tijdstip.

In figuren wordt de tijd steeds op de horizontale as uitgezet en de waarden $f(t)$ worden op een verticale "waarde"-as uitgezet. In onze tekst zijn deze waarden altijd eindige reële getallen.

Om inzicht te krijgen in $f(t)$ kunnen we deze functie beschrijven als een lineaire combinatie van "bouwsteentjes". We ontbinden $f(t)$ in een basis en dit noemen we de "analyse" van $f(t)$. Tevoren hebben we in vergelijking (1.12) gezien hoe een functie kan ontbonden worden in een orthonormale basis $\{e_n\}_{n \in \mathbb{N}}$. De formule wordt hier herhaald:

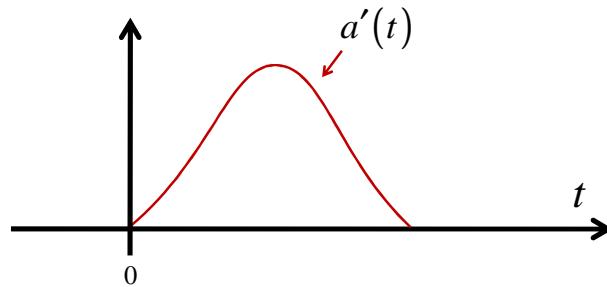
$$f = \sum_{n=0}^{\infty} \lambda_n e_n \quad (2.1)$$

De orthonormale basis bestaat dus uit alle functies e_n . Het is de bedoeling dat alle e_n samen het signaal $f(t)$ beschrijven, en elke e_n afzonderlijk vormt een "kenmerk" (karakteristieke component) van het signaal. De moeilijkheid van de analyse van het signaal is het ontdekken van welke e_n feitelijk moeten gebruikt worden om $f(t)$ te beschrijven. Maar wanneer dit eenmaal bepaald is, kunnen we het oorspronkelijke signaal $f(t)$ begrijpen en bewerken in functie van deze karakteristieke componenten e_n .

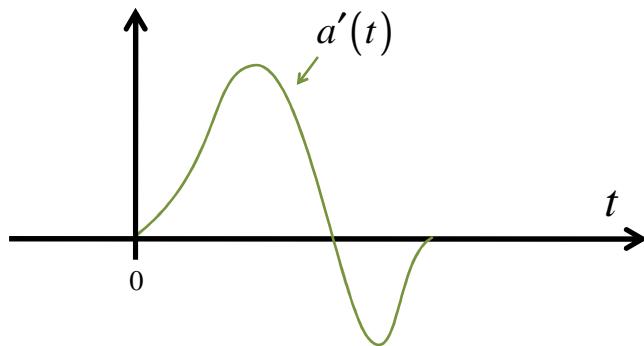
Elke coëfficiënt λ_n in de bovenstaande vergelijking geeft aan welke amplitude elk kenmerk e_n heeft aangenomen in signaal $f(t)$. De verzameling van deze λ_n noemen we de "expansion" van $f(t)$ in de basis $\{e_n\}_{n \in \mathbb{N}}$. Het komt er op neer dat we een soort van *projectie* maken van $f(t)$ naar de basisvectoren e_n .

De bovenstaande beschrijving van het vinden van de coëfficiënten λ_n in een orthonormale basis is wat er in essentie gebeurt bij *orthonormale* wavelet analyse. Men zal daar op voorhand vastleggen welke verschillende orthonormale basisvectoren e_n er mogen gebruikt worden om het signaal $f(t)$ in te ontbinden.

Veronderstel even een moeder "bultfunctie" $a(t) \in L^2(\mathbb{R})$ die bijna overal op de tijd-as nul is, behalve op een klein aaneengesloten stukje, m.a.w. met compacte support. Zie bijvoorbeeld de functie in Figuur 4. We kunnen natuurlijk ook een ingewikkeldere moeder bultfunctie $a'(t) \in L^2(\mathbb{R})$ beschouwen, ook met compacte support, zoals getoond in Figuur 5. Het precieze support-interval van beide bultfuncties is momenteel niet van belang. Bovendien is het in het algemeen niet verplicht dat $a(t)$ overal positief is. Dat is hier enkel gedaan om een eenvoudigere figuur te bekomen.



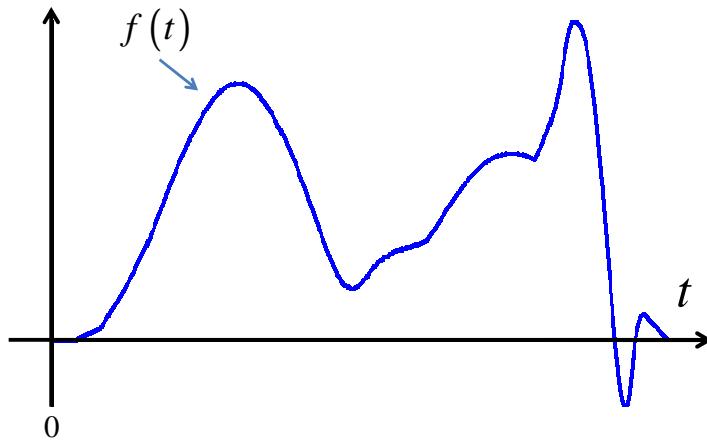
Figuur 4: moeder bultfunctie $a(t)$



Figuur 5: moeder bultfunctie $a'(t)$ met ook negatieve waarden.

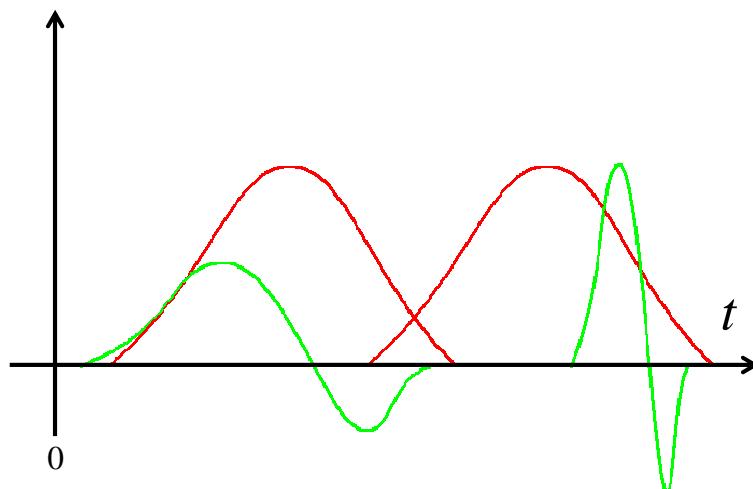
In paragraaf 1.1 hebben we gezien dat we functies dunner of breder kunnen maken (horizontaal schaleren) en ook horizontaal verschuiven over de tijd-as. Dit kunnen we ook toepassen op functies $a(t)$ en $a'(t)$. Veronderstel dat bepaalde horizontale schaleringen en horizontale verschuivingen van de moederfuncties $a(t)$ en $a'(t)$ een orthonormale basis $\{e_n\}_{n \in \mathbb{N}}$ opleveren.

In Figuur 6 is weergegeven welk signaal $f(t)$ we zullen beschouwen. Zoals de lezer kan zien heeft $f(t)$ zelf ook compacte support.

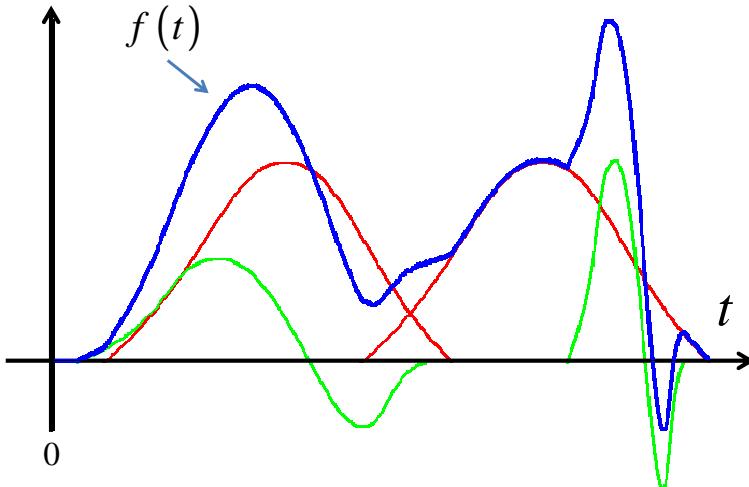


Figuur 6: signaalfunctie $f(t)$

De lineaire combinatie in vergelijking (2.1) drukt uit dat we ons signaal $f(t)$ gaan uitdrukken als een som van enkele bulfuncties in basis $\{e_n\}_{n \in \mathbb{N}}$, natuurlijk verticaal geschaleerd met een bijhorende λ_n . De verticaal geschaleerde basisvectoren zijn apart gevisualiseerd in Figuur 7 en samen met het signaal in Figuur 8. Elke λ_n in (2.1) rekent een e_n verticaal uit en maakt deze dus hoger of lager.



Figuur 7: gebruikte basisvectoren om het signaal in Figuur 6 te beschrijven.



Figuur 8: signaal $f(t)$ getoond samen met zijn ontbinding.

Zoals Figuur 7 weergeeft mogen de gebruikte bulten elkaar overlappen in de tijd en waar dat gebeurt moeten we hun functiewaarden optellen. We moeten op deze manier precies genoeg bulten verzamelen zodat op elke tijd t de som van de waarden van de bultfuncties de waarde van het signaal $f(t)$ aanneemt. Als we het signaal volledig beschreven hebben met de bultfuncties, dan is de analyse van het signaal $f(t)$ achter de rug.

We houden dan als analysedata over welke schaleringen we hebben toegepast op de bulten en op welke plaatsen in de tijd we deze bulten hebben gepositioneerd (horizontaal verschoven). Deze analysedata is dus een lijst van kenmerken van het signaal. Elk signaal zal zijn eigen lijstje hebben omdat voor elk signaal de bulten weer anders moeten geschaleerd en verschoven worden volgens de tijd-as. Bovendien is het lijstje van kenmerken ook verschillend wanneer andere bultfuncties gebruikt worden!

Je ziet ook al intuïtief dat brede bulten de eerder traag variërende kenmerken van het signaal beschrijven, dus de grote opvallende bewegingen van de functiewaarden. De dunneren bulten moeten bij deze grote bulten worden opgeteld en beschrijven de hoogfrequente delen van het signaal, dus de kleinere "grillen" van de functiewaarden van $f(t)$. Het ontdekken van traag variërende kenmerken van een signaal noemt men het toepassen van een *lowpass filter* op dat signaal. Daartegenover staat het ontdekken van de hoogfrequente componenten van een signaal, wat gebeurt met een *highpass filter* (Strang & Nguyen, Wavelets and Filter Banks, 1996). Hierover volgt later meer uitleg.

Voor deze voorbeeldontbinding van Figuur 7 zijn alle coëfficiënten λ_n uit (2.1) positief genomen. Het is natuurlijk ook mogelijk om negatieve coëfficiënten λ_n te gebruiken waardoor de bijhorende basisvector e_n dan gespiegeld wordt t.o.v. de tijd-as. Het was in dit voorbeeld niet nodig om negatieve λ_n te gebruiken om de negatieve waarden van $f(t)$ te modelleren. Deze zijn gemodelleerd met de negatieve waarden van een geschaleerde versie van moeder bultfunctie $a'(t)$.

Nog een laatste opmerking: het is geen toeval dat we hier precies twee moeder bultfuncties $a(t)$ en $a'(t)$ hebben gebruikt. Bij de wavelet-analyse die we in deze tekst zullen bespreken worden ook slechts twee moeder bultfuncties gebruikt. De resultaten die men bekomt bij wavelet-analyse zullen afhangen van welke vormen we vermoeden dat ze het signaal goed beschrijven. We gaan als het ware bepaalde vormen in het signaal proberen in te passen om op die manier het signaal te benaderen.

3. Multiresolutie

In vorig hoofdstuk hebben we kort gesproken over het bekijken van een signaal op meerdere niveaus van detail. Dit noemt men "multiresolutie". In dit hoofdstuk wordt aangegeven hoe de wavelet theorie dit concept ondersteunt.

3.1. Maak functies met "Scaling functions"

Veronderstel een functie $\varphi(t) \in L^2(\mathbb{R})$. In de literatuur wordt deze de "scaling function" genoemd, maar misschien was de naam "scaled function" iets duidelijker geweest omdat deze functie $\varphi(t)$ horizontaal wordt samengedrukt en uitgerokken. Maar om consistentie te behouden met de literatuur gebruiken we de term scaling function.

Men mag zich deze scaling function $\varphi(t)$ nu voorstellen als de bulten uit hoofdstuk 2.3, dus met een compact support. De lezer kan al eens een blik werpen op het linkerpaneel van Figuur 11 en Figuur 12 om een idee te krijgen van hoe scaling functions er kunnen uitzien.

Zoals we in paragraaf 1.1 hebben gezien, kunnen we deze functie horizontal verschuiven (volgens de tijd-as) als volgt:

$$\varphi_k(t) = \varphi(t-k) \quad k \in \mathbb{Z}$$

We kunnen deze functie ook schalen door haar horizontaal (volgens de tijd-as) samen te drukken als volgt:

$$\varphi_j(t) = \varphi(2^j t) \quad j \in \mathbb{Z}$$

Hierbij is 2^j de mate waarin de functie horizontaal wordt samengedrukt. Deze wordt de "scaling factor" genoemd. We geven enkele concrete voorbeelden:

- | | |
|--|--|
| 1) $j=0 \Rightarrow \varphi_0(t) = \varphi(t)$ | $\varphi_0(t)$ is precies gelijk aan $\varphi(t)$ |
| 2) $j=1 \Rightarrow \varphi_1(t) = \varphi(2t)$ | $\varphi_1(t)$ is 2 keer minder breed als $\varphi(t)$ |
| 3) $j=3 \Rightarrow \varphi_3(t) = \varphi(2^3 t) = \varphi(8t)$ | $\varphi_3(t)$ is 8 keer minder breed als $\varphi(t)$ |

In het algemeen kan men een functie horizontaal samendrukken door i.p.v. 2^j een algemeen reëel of natuurlijk getal te gebruiken. Maar in de context van eerste generatie wavelets worden deze schalingen meestal gedaan met machten van 2. Ook andere machten kunnen in principe gebruikt worden, maar 2 is voldoende en ook het eenvoudigste (Daubechies, 1992).

Horizontaal verschuiven en schalen kan ook gecombineerd gebeuren, zodat we een nieuwe soort scaling functions bekomen die afstammen van $\varphi(t)$:

$$\varphi_{j,k}(t) = \varphi(2^j t - k) \quad j, k \in \mathbb{Z} \quad (3.1)$$

Hierbij is 2^j de horizontale scaling factor, en $k/2^j$ de translatie in t . Merk op dat in deze context j en k integers zijn. De scaling factor 2^j komt dus uit de rationale getallen wanneer een negatieve macht j gebruikt wordt, wat een uitrekking volgens de tijd-as inhoudt. Door de constructie in (3.1) geldt dat alle $\varphi_{j,k}(t)$ ook in $L^2(\mathbb{R})$ zitten, net zoals $\varphi(t)$.

Het volgende kadertje is heel belangrijk:

De scaling functions die we in deze tekst beschouwen zijn zodanig dat voor een vaste j alle $\varphi_{j,k}(t)$ orthogonaal op elkaar staan.

Bovendien, alle scaling functions die wij beschouwen hebben norm gelijk aan één: $\|\varphi\| = 1$.

Door de "moeder"-functie $\varphi(t)$ dus te verschuiven en horizontaal samen te drukken, en lineaire combinaties te nemen van de resulterende functies $\varphi_{j,k}(t)$, kunnen we nieuwe functies samenstellen als volgt:

$$f(t) = \sum_{k \in \mathbb{Z}} a_k \varphi_{j,k}(t) \quad \text{met } a_k \in \mathbb{R} \quad (3.2)$$

Merk op dat de scaling factor j vast is, hier komen we later op terug. De verzameling van al dergelijke functies $f(t)$ wordt het "span" van de verzameling $\{\varphi_{j,k}(t)\}$ genoemd. Je kan de verzameling van al die $f(t)$ immers bekijken als voortgebracht of "opgespannen" door alle $\varphi_{j,k}(t)$. Indien we duidelijk willen aangeven dat de coëfficiënten in de sommatie horen bij een bepaalde j , zullen we $a_{j,k}$ noteren i.p.v. a_k .

De sommatie in gelijkheid (3.2) is een concrete vorm van die in gelijkheid (1.12). De $\varphi_{j,k}(t)$ komen overeen met de e_n en de a_k komen overeen met de λ_n .

Even terzijde, in paragraaf 1.2 hadden we reeds opgemerkt dat $n \in \mathbb{N}$ gewoon een identificatienummer is voor elke basisvector e_n . In gelijkheid (3.2) speelt $k \in \mathbb{Z}$ een beetje deze rol, want het is mogelijk om een ordening op de basisvectoren $\varphi_{j,k}(t)$ te plaatsen door de integers \mathbb{Z} te mappen naar de natuurlijke getallen. Dit is echter niet zo belangrijk.

De sommatie in (3.2) is theoretisch gezien oneindig indien de functie $f(t)$ zich bijvoorbeeld uitstrekkt naar $-\infty$ of naar ∞ (of naar beiden). We maken wel de veronderstelling dat voor een gegeven tijdstip t en een vaste j er slechts een eindig aantal van de scaling functions $\varphi_{j,k}(t)$ overlappen waardoor een functiewaarde $f(t)$ nooit oneindig groot kan worden:

$$\forall t : \{k \mid \varphi_{j,k}(t) \neq 0\} \text{ is eindig}$$

Door een goede keuze van de functie $\varphi(t)$ zal aan deze eis voldaan zijn. Dit is het geval bij alle $\varphi(t)$ die we zullen beschouwen. Het onderzoek in Wavelet-theorie heeft zich onder andere gericht op het ontwerpen van de functies $\varphi(t)$ met deze en andere wenselijke eigenschappen, zoals ook de bovengenoemde orthonormaliteit.

Omdat $\forall j, k \in \mathbb{Z} : \varphi_{j,k}(t) \in L_2(\mathbb{R})$, geldt ook $f(t) \in L_2(\mathbb{R})$.

De schalering j verdient wat extra aandacht. Als $j > 0$, dan worden de $\varphi_{j,k}(t)$ horizontaal dunner dan $\varphi(t)$. Er kunnen met dundere scaling functions meer verschillende $f(t)$ gemaakt worden dan met bredere scaling functions. Dat is intuïtief duidelijk omdat door dundere scaling functions te gebruiken kan men fijner detail voorstellen in een gemaakte $f(t)$. Bredere scaling functions kunnen enkel grove vormen in het signaal modelleren. Men kan in het algemeen niet voor een vaste j alle detailniveaus "simuleren" door wat extra translaties uit te voeren van een scaling function en de daaruit resulterende $\varphi_{j,k}(t)$ op te tellen. Voor meer detail in de te construeren $f(t)$ ben je genoodzaakt een hogere j in te schakelen.

Er geldt ook: voor verschillende schaleringswaarden j_1 en j_2 en een vaste translatiewaarde k , waarbij $j_1 > j_2$, zal de translatie in $\varphi(2^{j_1}t - k)$ kleiner zijn dan in $\varphi(2^{j_2}t - k)$. De dundere scaling functions $\varphi_{j_1,k}(t)$ zullen dus ook fijnere horizontale stapjes maken dan de bredere functies $\varphi_{j_2,k}(t)$. Om dit in te zien moeten we ons herinneren wat in paragraaf 1.1 werd gezegd over de verschuiving. Voor j_1 zal de verschuiving naar rechts met stapgrootte $k/2^{j_1}$ gebeuren en voor j_2 zal deze stapgrootte $k/2^{j_2}$ zijn en dus groter dan de eerste.

De functie $\varphi(t) \in L_2(\mathbb{R})$ wordt dus de "scaling" function genoemd omdat we haar kunnen "scalen" om op die manier verschillende vormen van detail in functies $f(t)$ voor te stellen. Omdat de hoeveelheid detail in de functies $f(t)$ afhangt van de scaling factor j , zullen we het span van $\{\varphi_{j,k}(t)\}$ noteren als V_j .

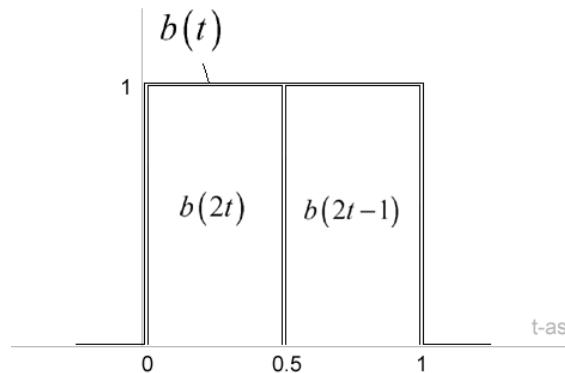
V_j is per definitie een vectorruimte waar alle functies $f(t)$ als hierboven inzitten. In een vectorruimte zijn ook basisvectoren aanwezig. Elke functie in een vectorruimte kan geschreven worden als een lineaire combinatie van de basisvectoren van die vectorruimte. In lineaire algebra zijn deze basisvectoren tuples en in deze context kunnen deze basisvectoren functies zijn. In V_j zijn de basisvectoren alle $\varphi_{j,k}(t)$. De basisvectoren van vectorruimte V_j zijn in dit geval horizontale verschuivingen van een geschaleerde scaling function. Elke vectorruimte V_j heeft een vaste j . Hierdoor is de breedte (schalering) van de basisvectoren per ruimte V_j vastgelegd. Hierdoor heeft V_j als basisvectoren horizontale verschuivingen van functies die steeds even breed zijn. Maar merk ook op dat de stapgrootten in deze horizontale verschuivingen *gehele* veelvouden moeten

zijn van $1/2^j$. Hierdoor zijn niet alle horizontale verschuivingen in een V_j beschikbaar. Hoe groter j , hoe dunner de $\varphi_{j,k}(t)$ zullen zijn en ook hoe "fijner" hun horizontale verschuivingen mogen zijn. Dit komt overeen met onze intuïtie: hoe dunner de basisvectoren, hoe meer detail zij kunnen voorstellen in een gesynthetiseerd signaal $f(t)$. En daartoe moeten we deze dunne basisvectoren ook preciezer kunnen plaatsen op de tijd-as, met kleinere stapgrootten. Voor $j_1 > j_2$ noemen we de functies $f(t)$ in V_{j_1} van *hogere resolutie* dan de functies in V_{j_2} .

Lagere resolutie scaling functies kunnen gemaakt worden door meerdere hogere resolutie scaling functies samen te nemen. We veronderstellen in deze tekst ook dat de scaling functions die wij beschouwen zodanig ontworpen zijn dat dit mogelijk is. Een voorbeeld hiervan kunnen we zien aan de hand van de "box"-function of "indicator"-function $b(t)$. Deze functie is constant over een eindig interval van de tijd-as en wordt meestal gedefinieerd als volgt:

$$b(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & \text{elders} \end{cases} \quad (3.3)$$

Dit is trouwens een standaard voorbeeld van een eenvoudige scaling function. In Figuur 9 is te zien dat 1 lage resolutie functie (de grote, brede functie) kan beschouwd worden als de som van twee hogere resolutie functies (de helft dunner) die op een gepaste wijze verschoven zijn.



Figuur 9: box-function $b(t)$ getoond als een samenstelling van kleinere box-functions $b(2t)$ en $b(2t-1)$.

We zien dat de lage resolutie moeder scaling function $b(t)$ een lineaire combinatie is van de twee hogere resolutie versies, zodat we mogen schrijven: $b(t) = 1 \cdot b(2t) + 1 \cdot b(2t-1)$. Tenslotte vermelden we nog dat de norm van de box-function één is. Dit kan men inzien als volgt, gebruik makend van (1.11):

$$\|b\|^2 = \int_{-\infty}^{\infty} |b(t)|^2 dt = \int_0^1 1 dt = 1 \Rightarrow \|b\| = 1$$

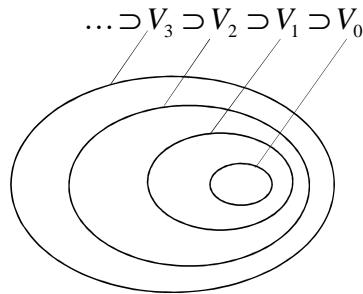
Deze box-function is maar een voorbeeld. Er bestaan ook andere scaling functions zodat de lage resolutie versies ervan steeds uitgedrukt kunnen worden als een lineaire combinatie van hogere resolutie versies van die scaling function. In het algemeen, mits de juiste scaling function gebruikt wordt, mogen we bijgevolg schrijven dat de lage resolutie vectorruimten V_j bevatten zijn in de hogere resolutie vectorruimten:

$$V_{-\infty} \subset \dots \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \dots \subset V_\infty$$

Dus $\forall j \in \mathbb{Z}: V_j \subset V_{j+1}$.

Naarmate j blijft toenemen, worden de basisvectoren in V_j steeds dunner. Zoals hierboven opgemerkt zullen we hierdoor met lineaire combinaties van de basisvectoren uit V_j steeds meer vormen in de gesynthetiseerde signalen $f(t)$ kunnen aanbrengen, zodat uiteindelijk de basis $\{\varphi_{j,k}(t)\}$ een basis is voor heel $L^2(\mathbb{R})$: $V_\infty = L^2(\mathbb{R})$.

Deze nesting van vectorruimten kunnen we grafisch weergeven, zoals in Figuur 10.



Figuur 10: nesting van de scaling function ruimten

Om even te herhalen: we genereren een familie van functies op basis van één moeder scaling function: per vectorruimte V_j worden de basisvectoren afgeleid van de moeder scaling function door deze te schalen met factor j en te verschuiven. Meestal wordt hier echter nog een amplitude bij vermenigvuldigd om ervoor te zorgen dat de norm van de gegenereerde functie gelijk is aan de norm van de moeder function. Dit hadden we tevoren niet genoteerd, maar in het algemeen moeten we schrijven (Burrus, Gopinath, & Guo, 1998):

$$\varphi_{j,k}(t) = 2^{j/2} \varphi(2^j t - k) \quad (3.4)$$

Doordat een samengedrukte scaling function een kleinere support heeft, moeten we hem ook even verticaal uitrekken met een extra factor $2^{j/2}$. Als de support kleiner gemaakt wordt met factor 2^j , dan maken we dus de hoogte gewoon groter met factor $2^{j/2}$ waardoor de norm gelijk blijft. Vanaf nu veronderstellen we voor $\varphi_{j,k}(t)$ steeds uitdrukking (3.4).

Zoals tevoren reeds opgemerkt, mag de lezer veronderstellen dat alle scaling functions die we in

deze tekst behandelen zodanig ontworpen zijn dat hun norm één is. Hierdoor heeft elke $\varphi_{j,k}(t)$ ook norm één. Zie ook de eigenschap hieronder. Bijgevolg heeft elke vectorruimte V_j een orthonormale basis en dus ook $L^2(\mathbb{R})$.

Eigenschap: indien geldt $\|\varphi(t)\| = 1$, dan heeft $2^{j/2} \varphi(2^j t - k)$ ook norm één.

Bewijs:

Herinner u dat de norm $\|\varphi_{j,k}(t)\|$ gedefinieerd is als $\sqrt{\langle \varphi_{j,k}(t), \varphi_{j,k}(t) \rangle}$. Voor eenenvoudigere schrijfwijze stellen we $a = 2^j$ (a is dus een positief getal). We laten eerst de voorfactor $2^{j/2}$ weg en berekenen volgende integraal:

$$\int_{-\infty}^{\infty} \varphi^2(at - k) dt$$

Voer de volgende substitutie door:

$$\begin{aligned} u &= at - k \\ \Rightarrow du &= u' dt = a dt \\ \Leftrightarrow \frac{1}{a} du &= dt \end{aligned}$$

Er geldt, omdat $a > 0$:

$$\begin{aligned} t \rightarrow -\infty &\Rightarrow u \rightarrow -\infty \\ t \rightarrow \infty &\Rightarrow u \rightarrow \infty \end{aligned}$$

En verder:

$$\int_{-\infty}^{\infty} \varphi^2(at - k) dt = \frac{1}{a} \int_{-\infty}^{\infty} \varphi^2(u) du = \frac{1}{a}$$

Deze laatste stap mag omdat $\|\varphi(t)\| = 1$. Uit bovenstaand resultaat leiden we af:

$$a \int_{-\infty}^{\infty} \varphi^2(at - k) dt = 1 \Rightarrow \int_{-\infty}^{\infty} (\sqrt{a} \varphi(at - k))^2 dt$$

Wanneer we terug $a = 2^j$ stellen, leiden we af dat $\|2^{j/2} \varphi(2^j t - k)\| = 1$. Q.E.D.

In het box-function voorbeeld hebben we om Figuur 9 duidelijker te houden de normalisatie van (3.4) ook nog niet in rekening gebracht. Maar door deze normalisatie met factor $2^{j/2}$ worden de dunneren box-functions in feite ook hoger dan in Figuur 9 werd weergegeven. De verhogingsfactor

is $\sqrt{2}$ omdat j in dat voorbeeld één is. We tonen eerst kort aan dat de norm van $\sqrt{2}b(2t)$ gelijk is aan de norm van $b(t)$:

$$\|\sqrt{2}b(2t)\|^2 = \int_{-\infty}^{\infty} |\sqrt{2}b(2t)|^2 dt = 2 \int_0^{1/2} 1 dt = 2 \cdot \frac{1}{2} = 1 \Rightarrow \|\sqrt{2}b(2t)\| = 1$$

De coëfficiënten uit de lineaire combinatie moeten dus kleiner worden om de vermenigvuldiging met $\sqrt{2}$ te compenseren:

$$b(t) = \frac{1}{\sqrt{2}}\sqrt{2}b(2t) + \frac{1}{\sqrt{2}}\sqrt{2}b(2t-1)$$

Het verband dat een lagere resolutie scaling function kan opgebouwd worden uit meerdere hogere resolutie scaling functions drukken we het eenvoudigst uit via de moeder scaling function: de moeder scaling function $\varphi(t)$ (lagere resolutie) is een lineaire combinatie van enkele scaling functions $\varphi_{1,k}(t)$ op één hoger resolutie niveau (de helft dunner dan de moeder).

The dilation equation (3.5)

$$\begin{aligned}\varphi(t) &= \sum_{k=0}^{N-1} h(k) \varphi_{1,k}(t) \\ &= \sum_{k=0}^{N-1} h(k) \sqrt{2} \varphi(2t-k) \\ &= \sqrt{2} \sum_{k=0}^{N-1} h(k) \varphi(2t-k)\end{aligned}$$

Hierbij is $N \in \mathbb{N}$ een even getal.

In het Engels zijn er ook andere veel gebruikte namen voor deze vergelijking:

- the refinement equation
- the multiresolution analysis (MRA) equation

De $h(k)$ noemen we de *scaling function coëfficiënten*. Deze coëfficiënten drukken uit dat de lagere resolutie $\varphi(t)$ kan uitgedrukt worden als een lineaire combinatie van verschoven $\sqrt{2}\varphi(2t)$ op 1 resolutie-niveau hoger. Deze hogere resolutie functies vormen samen immers een orthonormale basis waarin $\varphi(t)$ kan uitgedrukt worden. Merk op dat coëfficiënt $h(k)$ hoort bij een $\varphi_{1,k}(t)$, die zelf een verschuiving van $k/2$ naar rechts is van $\varphi(2t)$.

De verzameling coëfficiënten $h(k)$ is eindig wanneer de scaling functions compact support hebben. Zoals in paragraaf 2.1 vermeld werd, veronderstellen wij steeds scaling functions (en later ook wavelets) met compact support, vandaar schrijven we een eindige sommatie in (3.5). De scaling functions die we in deze tekst beschouwen hebben steeds een even aantal scaling function

coëfficiënten: $h(0), \dots, h(N-1)$ met N een even getal. Een scaling function met N scaling function coëfficiënten heeft support $[0, N-1]$ (Strang & Nguyen, Wavelets and Filter Banks, 1996). In een later hoofdstuk zullen we zien dat de coëfficiënten $h(k)$ gebruikt worden in een filter bank systeem.

De scaling functions moeten zodanig ontworpen worden dat $V_j \subset V_{j+1}$. De dilation equation is dan een direct gevolg van het feit $V_0 \subset V_1$.

De precieze waarden die $h(k)$ en k moeten aannemen hangen af van welke scaling function men beschouwt. Bij het box-function voorbeeld geldt dus dat $N = 2$. Concreet zijn de scaling function coëfficiënten als volgt: $h(0) = 1/\sqrt{2}$ en $h(1) = 1/\sqrt{2}$. Volgens de algemene formule van de dilation equation (3.5) schrijft men dit voorbeeld dus als:

$$\varphi(t) = \frac{1}{\sqrt{2}}\varphi(2t) + \frac{1}{\sqrt{2}}\varphi(2t-1)$$

Tenslotte merken we op de sommatie in de dilation equation (3.5) een speciaal geval is van de sommatie in (1.12): de sommatie in de dilation equation is echter niet oneindig zoals in (1.12), maar eindig. Als $\varphi(t) = \sum_{k=0}^N h(k)\varphi_{1,k}(t)$, dan moet gelden:

$$\begin{aligned} h(k) &= \langle \varphi(t), \varphi_{1,k}(t) \rangle \\ &= \sqrt{2} \int_{-\infty}^{\infty} \varphi(t) \varphi(2t-k) dt \end{aligned}$$

3.2. In de praktijk

We willen een signaal $f(t)$ ontbinden in kenmerkende deelsignalen. Dit gebeurt via het algoritme van wavelet-analyse. Andere namen zijn wavelet decompositie of wavelet-transformatie. Één iteratie van het wavelet-analyse-algoritme zal een gesampled signaal s als input ontvangen, bijvoorbeeld in de vorm van een array van floating-point getallen. Nu komt een zeer belangrijk punt: we doen alsof de samples in het discrete signaal s de coëfficiënten zijn bij de basisvectoren van een bepaalde vectorruimte V_j . Waarom mag dit? Het lijkt verkeerd om concrete samplewaarden te bekijken als coëfficiënten die horen bij basisvectoren. Voor een resolutie-niveau j dat hoog genoeg ligt, zullen de scaling functions in de bijhorende vectorruimte V_j zich gedragen als zeer dunne *impulsen*. Het inner product van een continu signaal $f(t)$ met zo'n impuls-basisvector zal de coëfficiënt opleveren die bij die impuls-basisvector hoort. Maar omdat de impuls-basisvector erg dun is, komt dit eigenlijk neer op een sampling van het continue signaal $f(t)$. Indien de basisvectoren dun genoeg zijn zal de resulterende rij coëfficiënten mogen beschouwd worden als samples die vrij goed het verloop van het signaal beschrijven (Burris, Gopinath, & Guo, 1998).

In de praktijk willen we wavelet-analyse uitvoeren op bitmap-data en arrays van floating-point waarden. Deze zijn niet altijd afkomstig van het samplen van een continu fenomeen, maar worden vaak volledig artificieel gegenereerd door de computer. Met bovenstaande uitleg over impulsen kunnen we wel doen alsof onze artificiële data afkomstig is van een continu fenomeen, waarbij we doen alsof de waarden in de arrays de coëfficiënten zijn die horen bij de impulsen.

Vanaf nu kunnen we dus de discrete reeks waarden die we willen analyseren bekijken als de coëfficiënten die horen bij de basisvectoren van een vectorruimte V_j . Voor de eenvoud van de theorie en latere afleidingen zullen we het discrete signaal wel nog noteren met een continue functie $f(t)$. Dit wordt gedaan omdat de wiskundige theorie van wavelets in de literatuur vooral met continue functies wordt ontwikkeld en beschreven (Daubechies, 1992), (Mallat, 1989), (Mallat, 1999).

3.3. Multiresolutie Approximatie van functies

We veronderstellen een projectie die een signaalfunctie $f(t) \in L_2(\mathbb{R})$ benadert op resolutie j .

Het resultaat van deze projectie (benadering), genoteerd $f_j(t)$, bevindt zich in de ruimte V_j .

Door deze constructie is $f_j(t)$ een lineaire combinatie zijn de basisvectoren $\varphi_{j,k}(t)$ van V_j . De vectorruimte V_j kan geïnterpreteerd worden als de verzameling van alle mogelijke benaderingen op resolutie j van functies $f(t) \in L_2(\mathbb{R})$.

Beschouw volgende definitie:

Definitie 3: Multiresolutie Approximatie (benadering)
--

Een sequentie $\{V_j\}_{j \in \mathbb{Z}}$ van gesloten deelruimten van $L_2(\mathbb{R})$ is een multiresolutie approximatie (van $L_2(\mathbb{R})$) indien aan de volgende voorwaarden wordt voldaan (Mallat, 1989), (Mallat, 1999):

- 1) $\forall j, k \in \mathbb{Z} : g(t) \in V_j \Leftrightarrow g(t - k/2^j) \in V_j$
- 2) $\forall j \in \mathbb{Z} : V_{j-1} \subset V_j$
- 3) $\forall j \in \mathbb{Z} : g(t) \in V_j \Leftrightarrow g(t/2) \in V_{j-1}$
- 4) $\lim_{j \rightarrow -\infty} V_j = \bigcap_{j=-\infty}^{\infty} V_j = \{0\}$
- 5) $\lim_{j \rightarrow \infty} V_j = \text{Closure} \left(\bigcup_{j=-\infty}^{\infty} V_j \right) = L_2(\mathbb{R})$

Laten we een intuïtieve uitleg geven bij deze wiskundige eigenschappen:

- 1) De vectorruimte V_j is *invariant* onder translaties die veelvouden zijn van $1/2^j$. Dit betekent dat indien $g(t) \in V_j$, de functie $g(t - k/2^j) \in V_j$. Dit klopt omdat de basisvectoren $\varphi_{j,k}(t)$ van V_j zelf horizontale verschuivingen zijn van $\varphi(2^j t)$ waarbij de stapgrootten veelvouden zijn van $1/2^j$.
- 2) De benadering $f_j(t)$ (op resolutie j) bevat alle informatie om het oorspronkelijke signaal $f(t)$ te benaderen op de lagere resolute $j-1$.
- 3) Indien we een functie $g(t) \in V_j$ horizontaal uitrekken met factor twee, zullen zijn details ook met een factor 2 vergroot worden. De vergrote details zullen dan kunnen beschreven worden met de basisfuncties uit de lagere-resolutie ruimte V_{j-1} .
- 4) Omdat de basisvectoren in V_j niet alle details van de oorspronkelijke $f(t)$ kunnen beschrijven, zal $f_j(t)$ slechts een benadering zijn van $f(t)$. Indien we steeds ruwere basisvectoren gebruiken (steeds kleinere j), dan zal uiteindelijk $f_j(t)$ geen informatie meer bevatten van $f(t)$ en de nulfunctie zijn.
- 5) Wanneer de resolutie blijft toenemen, zullen de basisvectoren in V_j steeds meer detail van de oorspronkelijke $f(t)$ kunnen beschrijven en zal de benadering $f_j(t)$ steeds dichter bij $f(t)$ liggen. Dit noemt men *compleetheid*.

Het zal wegens bovenstaande eisen de bedoeling zijn dat $f_j(t)$ niet meer verandert indien we nogmaals de projectieoperator zouden uitvoeren op $f_j(t)$ zelf.

Bovendien, van alle functies in de ruimte V_j is $f_j(t)$ de functie die het meest "gelijkt" op $f(t)$. Dit kunnen we uitdrukken met de norm:

$$\forall g(t) \in V_j : \|g(t) - f(t)\| \geq \|f_j(t) - f(t)\|$$

De projectie van $f(t)$ is dus een *orthogonale projectie* op de vectorruimte V_j . In een later hoofdstuk zullen we een algoritme afleiden dat deze projectie zal kunnen uitvoeren.

In navolging van (3.2) kunnen we schrijven:

$$f_j(t) = \sum_{k \in \mathbb{Z}} a_k \varphi_{j,k}(t) \quad \text{met } a_k \in \mathbb{R}$$

In de praktijk zal men $f_j(t)$ dus gewoon kunnen uitdrukken als een sequentie coëfficiënten $(a_k)_{k \in \mathbb{Z}}$. Per tijdseenheid zijn er $1/2^j$ coëfficiënten, want de basisvectoren in V_j worden

verschoven met stapjes van grootte $1/2^j$. Zoals we hierboven hebben besproken zal wanneer de functie $f(t)$ naar rechts verschoven wordt met een stapgrootte l die een veelvoud is van $1/2^j$, zijn approximatie $f_j(t)$ ook verschuiven met stapgrootte l naar rechts. Dit komt erop neer dat er andere basisvectoren nodig zullen zijn om $f_j(t)$ te beschrijven, waardoor de sequentie $(a_k)_{k \in \mathbb{Z}}$ ook met stapgrootte l "naar rechts schuift". Dit wil zeggen dat er aan het begin van de sequentie l nullen worden bijgevoegd. In de praktijk zal men voor $f(t)$ een eindige sequentie samples nemen zodat de sequentie $(a_k)_{k \in \mathbb{Z}}$ ook eindig is.

Tenslotte, wat missen we hier nog? We hebben gezegd dat de verschillende benaderingen $f_j(t)$ niet meer de volledige informatie meer bevatten van de oorspronkelijke $f(t)$. We zijn bij echte toepassingen natuurlijk geïnteresseerd in verliesloze transformaties van de oorspronkelijke data. De volgende paragraaf introduceert een nieuw concept *wavelets* dat kan gebruikt worden om te praten over de detailinformatie tussen twee benaderingen $f_j(t)$ in.

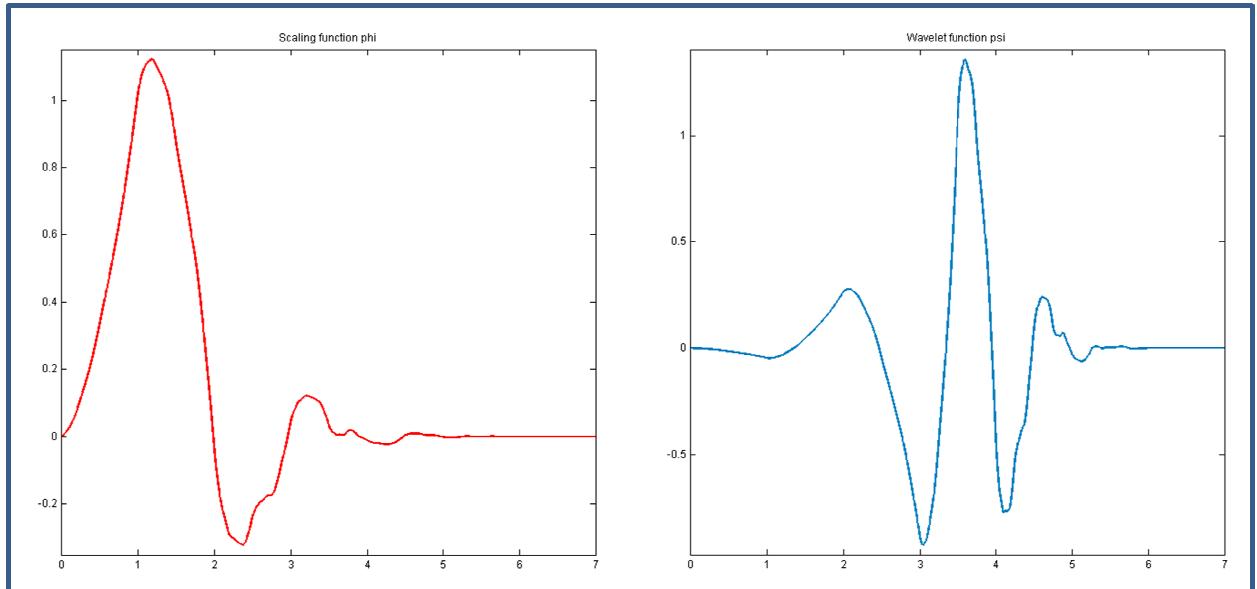
3.4. Wavelets

We gaan niet enkel de geschaleerde en verschoven scaling functions gebruiken om een signaal te beschrijven. We gaan ook een ander soort functies gebruiken, de "wavelets". Elke moeder scaling function heeft een bijhorende moeder wavelet. Bijvoorbeeld, de wavelet van de Box scaling function van tevoren heet de *Haar-wavelet*, genoemd naar de ontdekker ervan. Scaling functions hebben we in deze tekst met het symbool φ genoteerd en de wavelets worden in deze tekst met het symbool ψ genoteerd. Dit is consistent met de meeste literatuur.

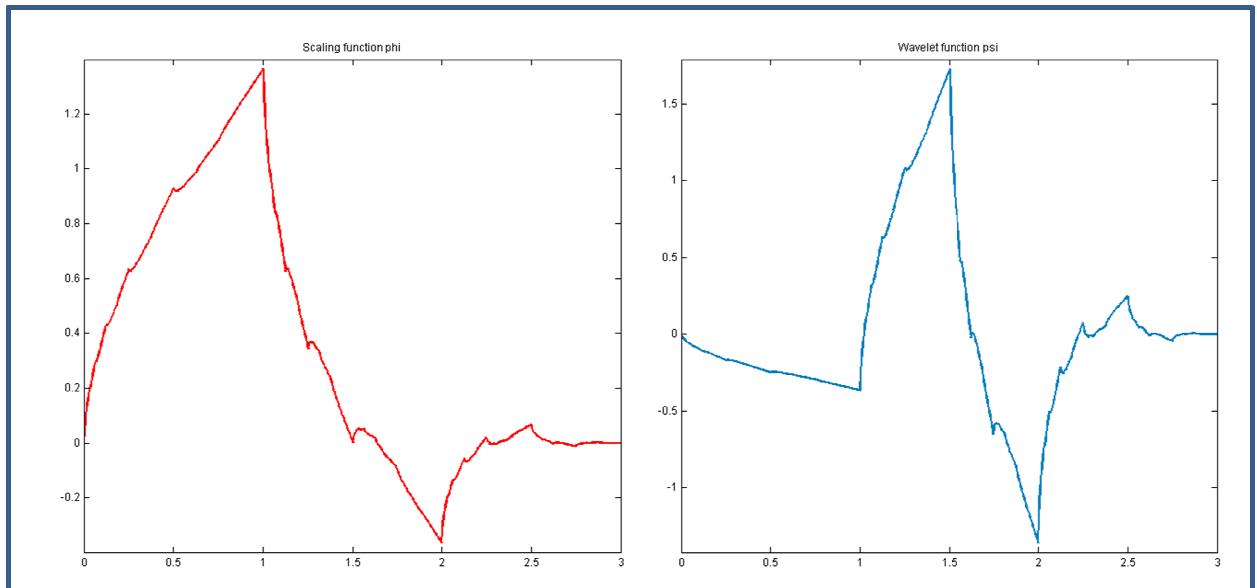
Wavelets kunnen ook, net zoals de scaling functions, voorkomen op een bepaalde resolutie j en horizontaal verschoven worden. Het enige verschil met een scaling function is gewoon hun vorm. Het is bij de wavelet functions de bedoeling dat hun vorm er net ietsje "ingewikkelder" uitziet dan de bijhorende scaling function. Het doel daarvan is dat de geschaleerde en verschoven scaling functions een ruwe beschrijving (approximatie) van een signaal $f(t)$ geven, maar hun partner-wavelets nodig hebben om meer details te kunnen modelleren van $f(t)$. De ingewikkeldere vorm van de wavelet komt doordat hij meer oscillaties maakt dan de bijhorende scaling function. Meer oscillaties duidt op hogere frequentie. In het begin van de tekst werd al even gesproken over lowpass en highpass filters. Scaling functions komen overeen met lowpass filters en wavelet functions komen overeen met highpass filters in de analyse. Hier komen we later nog op terug.

In Figuur 11 ziet men een illustratie van de Daubechies4 moeder scaling function en moeder wavelet en in Figuur 12 een illustratie van de Daubechies2 moeder scaling function en moeder wavelet. Men kan zien dat de wavelet telkens meer oscillaties vertoond dan de scaling function. In het Daubechies2 paar is dit minder uitgesproken. De getallen in de genoemde namen duiden op een bepaald lid van de Daubechies familie van orthonormale wavelet systemen. De orthonormale scaling functions en bijhorende wavelets van Ingrid Daubechies hebben geen expliciet voorschrijf, maar hun tekening kan geconstrueerd worden door de iteratie van een filterbank systeem (Burrus, Gopinath, & Guo, 1998),(Strang & Nguyen, Wavelets and Filter Banks, 1996). Filters worden nog besproken in een latere paragraaf, maar het construeren van de afbeeldingen van scaling functions en wavelets ligt buiten het bestek van deze tekst.

Opmerking: zoals men uit Figuur 11 en Figuur 12 kan afleiden, is het niet verplicht dat de scaling function steeds overal positieve waarden heeft.



Figuur 11: Daubechies Db4 scaling function en wavelet



Figuur 12: Daubechies Db2 scaling function en wavelet

Definitie 4: Orthogonaal Complement

Het orthogonaal complement B^\perp van een vectorruimte B in een vectorruimte A is gedefinieerd als:

$$B^\perp = \{x \in A : \forall y \in B \text{ geldt } \langle x, y \rangle = 0\}$$

Deze definitie komt neer op het feit dat een vectorruimte A in twee

deelruimten kan gesplitst worden, namelijk B^\perp en B , op zo'n manier dat alle elementen van B^\perp orthogonaal staan op alle elementen van B .
Notatie:

$$A = B \oplus B^\perp$$

Net zoals we verschillende vectorruimten V_j hebben waarvan de basisvectoren gemaakt worden door de moeder scaling function horizontaal te schaleren en te verschuiven, is het ook mogelijk om de moeder wavelet horizontaal te schaleren en te verschuiven waardoor basisvectoren van wavelet-vector-ruimten gemaakt kunnen worden. De basisvectoren van vectorruimten V_j die afgeleid zijn van de moeder scaling function zijn we scaling functions blijven noemen. Consistent met deze naamgeving noemen we de basisvectoren van de wavelet-vector-ruimten die afgeleid zijn van de moeder wavelet ook gewoon wavelets. De wavelet-vector-ruimte waar alle "wavelets van resolutie j " en hun lineaire combinaties inzitten noteren we met W_j . We definiëren W_j als het orthogonaal complement van V_j in V_{j+1} . Dit noteren we als volgt:

$$V_{j+1} = V_j \oplus W_j \quad (3.6)$$

Een wavelet die hoort bij resolutie j (dus $\in W_j$) en met $k/2^j$ verschoven is, noteren we als $\psi_{j,k}(t)$. Deze notatie is dus volledig analoog aan die van de scaling functions:

$$\psi_{j,k}(t) = 2^{j/2} \psi(2^j t - k)$$

De functies $g(t)$ in de vectorruimte W_j zijn lineaire combinaties van de $\psi_{j,k}(t)$ en worden als volgt uitgedrukt:

$$g(t) = \sum_{k \in \mathbb{Z}} b_k \psi_{j,k}(t) \quad \text{met } b_k \in \mathbb{R}$$

Indien we duidelijk willen maken dat de coëfficiënten bij een bepaalde resolutie j horen, schrijven we $b_{j,k}$ i.p.v. b_k .

De wavelets worden eigenlijk gemaakt uit de scaling functions. Er geldt wegens (3.6) immers dat de wavelets van resolutie j afkomstig zijn van de hogere resolutie vectorruimte V_{j+1} en dus feitelijk lineaire combinaties zijn van de scaling functions uit V_{j+1} . Dit geeft aan dat de wavelets van resolutie j dus fijner detail kunnen voorstellen dan de scaling functions van resolutie j . Belangrijk is dat de lezer beseft dat ondanks we " W_j " schrijven, en we spreken van de "wavelets op resolutie j ", we eigenlijk achterliggend altijd moeten onthouden dat W_j bestaat uit functies die net een iets hogere resolutie hebben dan de functies in V_j . We gebruiken toch " W_j " omdat we

voortaan de vectorruimten W_j en V_j beschouwen als een onlosmakelijk duo. De scaling functions en wavelets gaan we vanaf nu als 1 orthonormale basis beschouwen om een signaal te beschrijven. In 3.4.2 zullen we zien hoe de moeder wavelet concreet in verband staat met de moeder scaling function.

In de definitie van de vectorruimte W_j eisen we door de orthogonaal-complement-constructie dat de scaling functions en wavelets orthogonaal op elkaar staan. Dat wil zeggen dat hun inner product 0 is. Het is handig om te eisen dat W_j orthogonaal staat op V_j , want elke W_j staat hierdoor automatisch orthogonaal op alle W_k met $k < j$. Dit is juist omdat:

- Elke W_k is wegens constructie bevat in V_j
- W_j staat orthogonaal op V_j

Het is niet vereist dat de scaling functions van twee verschillende resolutie-niveaus loodrecht op elkaar staan, maar voor de wavelets is dat wel het geval. Het doel hiervan is dat bij de analyse een signaal $f(t)$ kan opgesplitst worden in één set coëfficiënten voor scaling functions en meerdere sets coëfficiënten van wavelets. De energie van de functie kan m.a.w. eenduidig verdeeld worden naar een orthonormale basis waar zowel scaling functions als wavelets inzitten. Er zullen geen meerdere verdelingen van de energie van de functie bestaan over deze basisvectoren, waardoor $f(t)$ uniek zal gerepresenteerd kunnen worden met zijn coëfficiënten. Dit verhaal is ook gekend in de context van een orthogonale basis in de lineaire algebra.

3.4.1. Wavelet-analyse

Om verder te gaan op de betekenis van wavelets, kunnen we de wavelet-ruimte W_j ook bekijken als een soort van "verschilruimte". Vanuit het perspectief van één individuele functie $f(t)$ zal een verschilfunctie kunnen gedefinieerd worden als volgt: $\Delta f_j(t) = f_{j+1}(t) - f_j(t)$. Deze verschilfunctie $\Delta f_j(t)$ is het verschil aan informatie tussen de projectie van $f(t)$ in de ruimte V_{j+1} en de projectie van $f(t)$ in de ruimte V_j . Dit is intuïtief gezien de fout die men maakt tussen benaderingen van $f(t)$ op twee nabijgelegen resoluties.

De wavelet ruimte W_j bevat wegens zijn definitie als orthogonaal complement alle mogelijke verschilfuncties tussen functies op resolutie-niveau j en functies op hoger resolutie-niveau $j+1$. Als we het vorige even omgekeerd bekijken, dan is elke functie $f_{j+1}(t)$ in V_{j+1} de som van twee delen: een *benadering* $f_j(t)$ van die functie in een lagere resolutie ruimte V_j en een bijhorende *detail-* of verschilfunctie $\Delta f_j(t)$ in W_j .

Omdat V_j en W_j orthogonale complementen van elkaar zijn in ruimte V_{j+1} kan men door lineaire combinaties te nemen van functies uit V_j en W_j nieuwe functies maken die fijnere details

bevatten dan de functies in V_j en W_j afzonderlijk. Deze functies met fijner detail bevinden zich dus in de ruimte V_{j+1} .

Het analyse algoritme zal in het licht van bovenstaande uitleg $f_j(t)$ ontbinden in twee lagere resolutie deelsignalen: $f_{j-1}(t)$ in V_{j-1} en $\Delta f_{j-1}(t)$ in W_{j-1} . Opgeteld geven deze twee deelsignalen terug $f_j(t)$.

We kunnen enkele concrete voorbeelden opschrijven van vergelijking (3.6):

$$\begin{aligned} V_1 &= V_0 \oplus W_0 \text{ en} \\ V_2 &= V_1 \oplus W_1 \end{aligned}$$

Er geldt daarom ook $V_2 = V_0 \oplus W_0 \oplus W_1$. We kunnen nu de ruimte $L_2(\mathbb{R})$ uitdrukken als:

$$L^2(\mathbb{R}) = V_0 \oplus W_0 \oplus W_1 \oplus W_2 \oplus \dots \quad (3.7)$$

Dit wil zeggen dat de ruimte $L_2(\mathbb{R})$ bestaat uit benaderingsfuncties afkomstig uit een lage resolutie vectorruimte V_0 , met daarbij opgeteld detailfuncties uit W_0, W_1, W_2, \dots . We mogen beginnen bij een willekeurige V_0 want in die ruimte zitten toch alle deelruimten V_j met $j < 0$. Maar natuurlijk kunnen we met de functies in V_0 niet alle detail uitdrukken van functies in ruimten V_j met $j > 0$. Om dat te doen hebben we dus de wavelet-ruimten nodig.

In de praktijk bepalen we bijvoorbeeld welk laagste resolutie-niveau V_b nog interessant is om de benadering van een signaal in te beschouwen. Tijdens het ontbinden van een signaal gaan we dan niet op een lagere resolutie dan dat niveau en we schrijven het signaal als een allerlaagste approximatie-signaal in V_b en daarbij ook detail-signalen in de wavelet-ruimten W_j met $j \geq b$.

Deze manier van werken komt dan overeen met (3.7), waarbij $V_0 = V_b$. Om een signaal op die manier te beschrijven, is het wel vereist dat we weten op welk resolutie-niveau J het inputsignaal zich bevindt want anders weet je niet wanneer het laagste resolutie-niveau van interesse bereikt is. Men kan eigenlijk ook maar detail-signalen hebben in de wavelet-ruimten W_j met $b \leq j < J$.

De detail-signalen in W_j met $j \geq J$ zijn nulfuncties, omdat het oorspronkelijke signaal geen informatie bevat op een resolutie hoger dan J .

In de praktijk is het daarnaast veel voorkomend dat men eigenlijk niet wil/kan expliciet maken op welk resolutie-niveau het inputsignaal zich bevindt. Men veronderstelt dan dat het te analyseren discrete signaal s zich bevindt op een bepaald (hoog) resolutie-niveau J , waarvan de precieze getalwaarde niet belangrijk is. Men zal in dat geval geen allerlaagste resolutie-niveau van interesse kunnen specifiëren, want er is geen referentiepunt. Dit komt overeen met onze opmerking in paragraaf 3.2.

Bij beide manieren van werken kan men in ieder geval nooit meer dan het maximaal aantal iteraties uitvoeren op het discrete invoersignaal. Het maximaal aantal iteraties van de wavelet decompositie is een logaritmische functie van het aantal oorspronkelijke invoersamples. Daarover volgt later meer uitleg. Hieronder wordt al kort beschreven hoe de decompositie in zijn werk gaat. Zoals uiteengezet aan het begin van deze paragraaf, zal men doen alsof een (fictief) continu signaal werd benaderd op een bepaald resolutie-niveau J en dat de samples in s de coëfficiënten zijn die horen bij de basisvectoren van V_J . Er zijn geen coëfficiënten gekend voor de basisvectoren in W_J , dus de $b_{J,k}$ worden als nul verondersteld. Men gaat in een eerste iteratie van het analyse-algoritme vanuit deze approximatie-coëfficiënten $a_{J,k}$ de approximatie-coëfficiënten $a_{J-1,k}$ en wavelet coëfficiënten $b_{J-1,k}$ berekenen op één resolutie-niveau lager ($J - 1$). In een tweede iteratie zal het algoritme vanuit de berekende approximatie-coëfficiënten $a_{J-1,k}$ de approximatie-coëfficiënten $a_{J-2,k}$ en wavelet coëfficiënten $b_{J-2,k}$ berekenen op nog een resolutie-niveau lager ($J - 2$). Na p iteraties van de wavelet-analyse komt men dan uit op een approximatie van het signaal in ruimte V_{J-p} en men weet dat al het detail is achtergebleven in de tegengekomen wavelet-ruimten (in de vorm van de coëfficiënten $b_{j,k}$ met $J - p \leq j < J$). We bekomen een ontbinding van het oorspronkelijke signaal in deelsignalen die zich bevinden in de ruimten:

$$V_{J-p}, W_{J-p}, \dots, W_{J-1}$$

zodat

$$V_J = V_{J-p} \oplus W_{J-p} \oplus \dots \oplus W_{J-1} \quad (3.8)$$

Bijvoorbeeld, na het uitvoeren van één iteratie ($p = 1$) krijgen we:

$$V_J = V_{J-1} \oplus W_{J-1}$$

Belangrijk is dat men in de "standaard" wavelet-analyse niet de wavelet-coëfficiënten verder ontbindt, maar enkel de approximatie-coëfficiënten. Indien men toch de wavelet-coëfficiënten wenst te ontbinden, dan spreekt men van *wavelet-packets*, maar die vallen buiten het bestek van deze tekst.

3.4.2. De wavelet vergelijking

Net zoals we in de dilation equation (3.5) hebben uitgedrukt dat de moeder scaling function een lineaire combinatie is van scaling functions op één resolutie-niveau hoger, kunnen we de moeder wavelet function ook uitdrukken als een lineaire combinatie van verschoven scaling functions van datzelfde hogere resolutie-niveau:

De wavelet vergelijking (3.9)

$$\begin{aligned}\psi(t) &= \sum_{k=0}^{N-1} h_1(k) \varphi_{1,k}(t) \\ &= \sqrt{2} \sum_{k=0}^{N-1} h_1(k) \varphi(2t - k)\end{aligned}$$

Hierbij is $N \in \mathbb{N}$ een even getal.

De $h_1(k)$ noemen we de *wavelet coëfficiënten* en deze zijn verschillend van de scaling function coëfficiënten. Het feit dat er zulke coëfficiënten $h_1(k)$ bestaan volgt uit het feit dat $W_0 \subset V_1$. Er geldt:

$$\begin{aligned}h_1(k) &= \langle \psi(t), \varphi_{1,k}(t) \rangle \\ &= \sqrt{2} \int_{-\infty}^{\infty} \psi(t) \varphi(2t - k) dt\end{aligned}$$

De eindige sommatie in de wavelet-vergelijking en de berekening van de wavelet coëfficiënten is nogmaals en duidelijke toepassing van de sommatie in (1.12), omdat ruimte V_1 een orthonormale basis heeft waarin $\psi(t)$ eenduidig kan uitgedrukt worden. De moeder wavelet wordt zadanig ontworpen dat zij evenveel wavelet coëfficiënten heeft als de bijhorende moeder scaling function scaling function coëfficiënten heeft. Men eist dat een aantal belangrijke voorwaarden voldaan zijn over het verband tussen de $h_1(k)$ en de $h(k)$. Men kan hieruit een formule afleiden om de coëfficiënten $h_1(k)$ te berekenen uit de coëfficiënten $h(k)$ van dilation equation (3.5) :

$$\begin{aligned}\forall k = 0 \dots N : \\ h_1(k) &= (-1)^k h(N-1-k)\end{aligned}\tag{3.10}$$

We kunnen zien dat de coëfficiënten $h_1(k)$ dezelfde waarden bevatten als de $h(k)$, maar dan in omgekeerde volgorde en met alternerend minteken. De uitdrukking $N-1$ in (3.10) is enkel om een juiste zero-based indexatie te doen in de coëfficiënten $h(k)$.

Voor meer details over het verband tussen de wavelet coëfficiënten en de scaling function coëfficiënten verwijzen we naar (Burrus, Gopinath, & Guo, 1998) en (Strang & Nguyen, 1996). Later zullen we zien dat (3.10) nuttig is bij de implementatie van wavelet-analyse en synthese.

Voor de Haar-wavelet hebben we twee van dergelijke $h_1(k)$ -coëfficiënten ($N = 2$):

$h_1(0) = 1/\sqrt{2}$ en $h_1(1) = -1/\sqrt{2}$. Het voorschrift van de Haar-wavelet is:

$$w(t) = \begin{cases} 1 & 0 \leq t < 0.5 \\ -1 & 0.5 \leq t < 1 \end{cases}$$

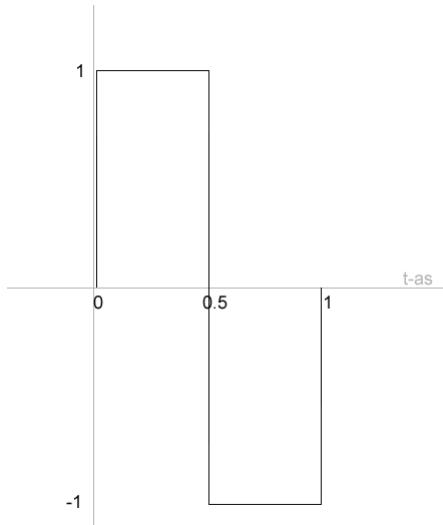
Als als we de Haar-wavelet schrijven volgens (3.9), dan plaatsen we de Box scaling function in de formule:

$$w(t) = \frac{1}{\sqrt{2}} b(2t) + \frac{-1}{\sqrt{2}} b(2t-1) = b(t)$$

We berekenen de norm van de Haar-wavelet:

$$\begin{aligned} \|b(2t) - b(2t-1)\|^2 &= \int_{-\infty}^{\infty} |b(2t) - b(2t-1)|^2 dt = \int_{-\infty}^{\infty} |0.5 - (-0.5)|^2 dt \\ &= \int_{-\infty}^{\infty} |1|^2 dt = \int_0^1 1 dt = 1 \end{aligned}$$

Deze norm is één, net zoals bij de Box scaling function. We zien hier een concreet bewijs dat de wavelets zelf ook een orthonormale basis vormen. In Figuur 13 is de Haar-wavelet weergegeven.



Figuur 13: de Haar-wavelet

Je kan zien dat vorm van de Haar-wavelet door de mintekens in de wavelet coëfficiënten nu "complexer" is dan die van de Box scaling function. Inderdaad, de meeste wavelets zien er iets ingewikkelder uit dan hun overeenkomstige scaling function omdat wavelets juist verantwoordelijk zijn voor het uitdrukken van extra detail. De Box-function wordt gekenmerkt door zijn constante waarde op een interval zodat hij lijkt op een baksteen. Bij de bijhorende Haar-wavelet (een lineaire combinatie van de Box-functions) zijn die horizontale stappen ook aanwezig. In het algemeen zal er wel niet zo'n duidelijk visueel verband te zien zijn tussen de moeder scaling function en de moeder wavelet.

4. Convolutie

De allereerste orthonormale scaling functions en wavelets die werden ontdekt hebben een expliciet voorschrift. Het nadeel van deze functies is het feit dat hun support oneindig was. De absolute waarde van deze functies neemt wel sterk af bij $t \rightarrow \pm\infty$, maar in de praktijk wil men graag scaling functions en wavelets hebben die *compact support* hebben zodat men deze als "zuiverdere" bouwstenen kan gebruiken die geen uitvloeiing meer hebben van zeer kleine functiewaarden op een oneindig interval.

De Vlaamse wiskundige Ingrid Daubechies heeft in 1988 ontdekt hoe men orthonormale families van scaling functions en bijhorende wavelets kan maken die compacte support hebben die verschillen van de Box scaling function en Haar wavelet. Analyse van een signaalfunctie met deze orthonormale families met compacte support kan gebeuren met een snelle implementatie gebaseerd op *Filter Banks*. Een Filter Bank is gewoon een verzameling digitale filters, afkomstig uit het domein van de digitale signaalverwerking. De filters dient men op het te verwerken signaal toe te passen om het signaal op te delen in deelsignalen. We zullen in hoofdstuk 5 onderzoeken hoe de theorie van wavelets aanleiding geeft tot een dergelijke implementatie.

Voor de duidelijkheid: *analyse* is het proces waarbij een inkomend signaal wordt opgesplitst in meerdere outputsignalen en *synthese* is het proces waarbij deelsignalen (terug) gecombineerd worden tot één outputsignaal. In deze tekst behandelen we enkel filter banks die bestaan uit 2 filters. Zowel bij analyse als bij synthese noemen we deze de *highpass* en *lowpass* filters.

In dit hoofdstuk zullen we eerst eventjes terug de wavelets vergeten en wat kennis aanhalen uit het domein van de digitale signaalverwerking. Wanneer we een signaal door een filter-bank laten passeren, zal er *convolutie* worden uitgevoerd van filters op dat signaal, met daarbij ook samplingoperaties. Het is zeer nuttig convolutie nu al te introduceren in de tekst zodat in hoofdstuk 5 de voorschriften van de convolutie-vormen kunnen herkend worden.

Dit hoofdstuk is gebaseerd op (Loy, 2007) en op (Strang & Nguyen, 1996), hoewel sommige details zelf werden aangevuld.

4.1. Inleiding tot convolutie

Convolutie is afkomstig van het Engelse woord *convolution*. Vrij vertaald betekent dit het "mengen" of door "elkaar kronkelen" van functies. In het discrete domein is dat het door elkaar kronkelen van arrays of sequenties. In de context van beeldverwerking is convolutie een elementsgewijze vermenigvuldiging van de waarden uit een filtermatrix met een submatrix (van dezelfde afmetingen) uit een pixel-afbeelding. Wij zullen enkel de één-dimensionale versie van convolutie bespreken waarbij we sequenties van waarden met elkaar vermenigvuldigen en geen matrices.

Zij f en g twee eindige sequenties van getallen, niet per se van dezelfde lengte. Het i^{de} sample van f en g noteren we met $f(i)$ respectievelijk $g(i)$. Met $|.|$ noteren we de lengte van een sequentie.

We definiëren sequentie h als het resultaat van convolutie van sequentie f met sequentie g , met lengte $|h| = |f| + |g| - 1$:

$$h = f * g$$

Hierbij is $*$ de binaire convolutie-operator die twee sequenties als operands gebruikt. We noemen h ook wel het *convolutie-product* van sequenties f en g . De elementen van h zijn gedefinieerd als volgt:

$$h(n) = \sum_{m=0}^n f(m)g(n-m) \quad (4.1)$$

Hierbij zijn m en n integers. De index n gaat van 0 tot en met $|h|-1 = |f|+|g|-2$. Telkens wanneer we in de sommatie van (4.1) een sample proberen op te vragen dat buiten het originele bereik van sequentie f of g ligt, negeren we de hele term in de sommatie waar dat gebeurt. We tellen in de sommatie dus enkel de termen $f(m)g(n-m)$ op wanneer beide samples $f(m)$ en $g(n-m)$ binnen het geldige bereik liggen van sequentie f respectievelijk sequentie g . De lengte van h is gelijk aan $|f|+|g|-1$ omdat we bij hogere waarde van n geen term $f(m)g(n-m)$ meer aantreffen in de sommatie waarvoor dit voldaan is.

In de praktijk kan men dit als volgt implementeren: als een index buiten het bereik van een sequentie ligt, dan nemen we aan dat de sequentie daar de waarde 0 heeft. Wanneer deze 0 dan vermenigvuldigd wordt met het andere sample in de termen van de sommatie, dan vallen die termen weg. In paragraaf 5.4 worden ook andere mogelijkheden aangereikt voor de implementatie van convolutie, maar dat is momenteel niet belangrijk.

De convolutie-operatie schuift dus een databasequentie g langs de filtersequentie f op een speciale manier. In de meeste toepassingen is de filter kleiner dan de databasequentie, maar dat is geen verplichting. Merk op dat bij de berekening van $h(n)$ in elke term van de sommatie aan de rechterkant de indices optellen tot n . Het toepassen van een filter f is een *tijdsonafhankelijke operator* omdat als we de indices van de databasequentie g zouden beschouwen als "tijd", de filter overal gelijk blijft bij het vermenigvuldigen met de datasamples. De waarden in de outputsequentie zijn lineaire combinaties van de waarden in de databasequentie g . De coëfficiënten van deze lineaire combinatie komen uit de filter f . De waarden in de filtersequentie worden ook *filtercoëfficiënten* genoemd.

We kunnen een voorbeeldje uitwerken zodat de werking gedemonstreerd wordt. Stel dat sequenties f en g beiden lengte 4 hebben. De geldige indices voor f en g liggen dan in $[0,3]$. Laten we de convolutie van f met g (beiden lengte 4) uitschrijven:

$$n=0 \quad h(0)=f(0)g(0)$$

$$n=1 \quad h(1)=f(0)g(1)+f(1)g(0)$$

$$n=2 \quad h(2)=f(0)g(2)+f(1)g(1)+f(2)g(0)$$

$$n=3 \quad h(3)=f(0)g(3)+f(1)g(2)+f(2)g(1)+f(3)g(0)$$

Naarmate n toeneemt, zal de convolutie steeds meer waarden in de sequenties f en g nodig hebben. Verder hebben we:

$$\begin{aligned} n=4 \quad h(4) \\ &= f(0)g(4) + f(1)g(3) + f(2)g(2) + f(3)g(1) + f(4)g(0) \\ &= f(0)\cdot 0 + f(1)g(3) + f(2)g(2) + f(3)g(1) + 0\cdot g(0) \\ &= f(1)g(3) + f(2)g(2) + f(3)g(1) \end{aligned}$$

$$\begin{aligned} n=5 \quad h(5) \\ &= f(0)g(5) + f(1)g(4) + f(2)g(3) + f(3)g(2) + f(4)g(1) + f(5)g(0) \\ &= f(0)\cdot 0 + f(1)\cdot 0 + f(2)g(3) + f(3)g(0) + 0\cdot g(1) + 0\cdot g(0) \\ &= f(2)g(3) + f(3)g(2) \end{aligned}$$

$$\begin{aligned} n=6 \quad h(6) \\ &= f(0)g(6) + f(1)g(5) + f(2)g(4) + f(3)g(3) + f(4)g(2) + f(5)g(1) + f(6)g(0) \\ &= f(0)\cdot 0 + f(1)\cdot 0 + f(2)\cdot 0 + f(3)g(3) + 0\cdot g(2) + 0\cdot g(1) + 0\cdot g(0) \\ &= f(3)g(3) \end{aligned}$$

$$\begin{aligned} n=7 \quad h(7) \\ &= f(0)g(7) + f(1)g(6) + f(2)g(5) + f(3)g(4) + f(4)g(3) + f(5)g(2) + \\ &\quad f(6)g(1) + f(7)g(0) \\ &= f(0)\cdot 0 + f(1)\cdot 0 + f(2)\cdot 0 + f(3)\cdot 0 + 0\cdot g(3) + 0\cdot g(2) + 0\cdot g(1) + 0\cdot g(0) \\ &= 0 \end{aligned}$$

$$n=8,9,10,\dots \quad h(n)=0$$

Na een tijdje zal de convolutie bij toenemende waarde van n alleen nog maar nulwaarden opleveren.

We kunnen vorig voorbeeld ook in een kolomvorm opschrijven. Hierbij zijn de termen uit de rechterleden van $h(n)$ in de rijen geschikt waarbij het rij-nummer (beginnende vanaf 0) overeenkomt met de index in de f -sequentie. Elke $h(n)$ is de som van zijn kolom boven de horizontale lijn.

$f(0)g(0)$	$f(0)g(1)$	$f(0)g(2)$	$f(0)g(3)$	
$f(1)g(0)$	$f(1)g(1)$	$f(1)g(2)$	$f(1)g(3)$	
	$f(2)g(0)$	$f(2)g(1)$	$f(2)g(2)$	$f(2)g(3)$
		$f(3)g(0)$	$f(3)g(1)$	$f(3)g(2)$
			$f(3)g(3)$	

$$\begin{array}{ccccccc} h(0) & h(1) & h(2) & h(3) & h(4) & h(5) & h(6) \end{array}$$

Deze kolomvorm is gebaseerd op de notatie voor vermenigvuldiging die gebruikt wordt bij het handmatig uitrekenen van een vermenigvuldiging. Alleen is er bij convolutie geen sprake van het overdragen van een rest-term tussen kolommen. Laten we deze "vermenigvuldigingsnotatie" eens illustreren op een concreet voorbeeldje met getallen. Stel:

$$\begin{aligned} g &= (4, 2, 3) \\ f &= (2, 5) \end{aligned}$$

Nu zullen we de convolutie van deze twee sequenties uitdrukken in vermenigvuldigingsnotatie:

$$\begin{array}{cccccc} g(2) & g(1) & g(0) & & 3 & 2 & 4 = g \\ f(1) & f(0) & & & 5 & 2 = f \\ \hline g(2)f(0) & g(1)f(0) & g(0)f(0) & & 6 & 4 & 8 \\ g(2)f(1) & g(1)f(1) & g(0)f(1) & & 15 & 10 & 20 \\ \hline h(3) & h(2) & h(1) & h(0) & 15 & 16 & 24 & 8 = h \end{array}$$

Belangrijk: de waarden in de betrokken sequenties worden in deze notatie in omgekeerde volgorde opgeschreven dan wanneer men ze gewoon opschrijft, waarbij de tijdindices van links naar rechts verlopen.

4.2. Eigenschappen van convolutie

Convolutie is commutatief. Laten we dit zonder verlies aan algemeenheid eens uitwerken op de filtersequentie $f = (a, b, c, d, \dots)$ en de datasequentie $g = (x, y, z, \dots)$:

$$\begin{array}{cccccc} & & \cdots & z & y & x \\ & & \cdots & c & b & a \\ \hline f * g : & & \cdots & az & ay & ax \\ & & \cdots & bz & by & bx \\ & & \cdots & cz & cy & cx \\ \hline & cz & bz + cy & az + by + cx & ay + bx & ax \end{array}$$

$$\begin{array}{r}
 & \cdots & c & b & a \\
 & \cdots & z & y & x \\
 \hline
 g * f : & \cdots & cx & bx & ax \\
 & \cdots & cy & by & ay \\
 & \cdots & bz & az & \\
 \hline
 & cz & bz + cy & az + by + cx & ay + bx & ax
 \end{array}$$

We kunnen zien dat beide convoluties hetzelfde resultaat opleveren.

Bovendien is convolutie ook een lineaire operator: de convolutie van een filter f met de som van twee databasequenties g_1 en g_2 is de som van de convoluties van f met g_1 en g_2 apart. Bovendien gaat de convolutie-operatie ook door een scalaire vermenigvuldiging van de databasequentie. Formeel genoteerd zijn beide lineaire eigenschappen:

$$\begin{aligned}
 (1) \quad f * (g_1 + g_2) &= f * g_1 + f * g_2 \\
 (2) \quad \forall s \in \mathbb{R} : f * (s \cdot g) &= s \cdot (f * g)
 \end{aligned}$$

Beide eigenschappen volgen rechtstreeks uit de definitie en ze zijn eenvoudig verifieerbaar door de convoluties uit te schrijven in de vermenigvuldigingsnotatie, zoals we hierboven voor de commutativiteits-eigenschap hebben gedaan.

4.3. Snelheidstrucje

In de vorige paragraaf hebben we gezien dat convolutie commutatief is. Je mag dus de rollen van f en g omdraaien in de sommatie.

In de praktijk kennen we natuurlijk de lengte van beide sequenties, en dit kunnen we uitbuiten tijdens het berekenen van de convolutie. Stel dat N de lengte is van filtersequentie f en dat $\forall i \in \mathbb{N}_0 : f(-i) \equiv 0$ en dat $\forall i \in \mathbb{N}, i \geq N : f(i) \equiv 0$. Hier stellen we dus dat alle filtercoëfficiënten buiten het originele bereik van de filter gelijk zijn aan nul. Over hoe de samples eruit zien buiten het originele bereik van de databasequentie zeggen we niets, hiervoor kan men paragraaf 5.4 raadplegen.

We kunnen twee convoluties definiëren die beiden hetzelfde resultaat opleveren als de convolutie van paragraaf 4.1:

$$h(n) = \sum_{m=0}^{N-1} f(m) g(n-m)$$

$$h(n) = \sum_{m=n-N+1}^n g(m) f(n-m)$$

Hierbij geldt $n = 0 \dots (|g| + N - 2)$, net zoals bij de allereerste definitie van convolutie in 4.1. De verschillen van deze vormen met de vorm in 4.1 zijn de grenzen bij de sommaties. Deze twee nieuwe sommatievormen berekenen slechts de strikt noodzakelijke samples. Dit mag omdat we anders toch buiten de grenzen van de filtersequentie treden en de termen dan per definitie gelijk zijn aan nul.

Deze paragraaf is maar een detail, gewoon om aan te geven dat we niet steeds $m = 0 \dots n$ moeten uitvoeren in de sommatie. Dit mag enkel indien we aannemen dat de filtercoëfficiënten buiten het bereik van de filter f gelijk zijn aan nul. We gaan in latere paragrafen niet explicet meer stilstaan bij deze efficiëntere implementatie, maar zullen de gewone vorm uit paragraaf 4.1 veronderstellen wanneer we nadenken over het gebruik van convolutie in een bepaalde context.

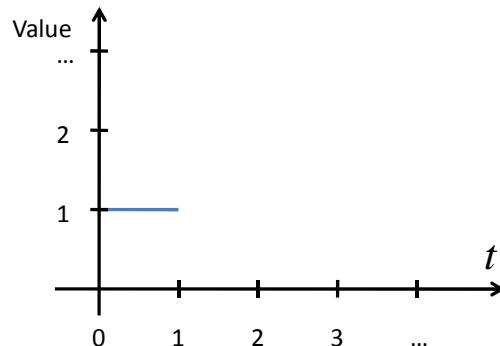
4.4. Betekenis van convolutie

Er is een speciaal geval van inputsequentie g , namelijk de *impuls* δ . Deze sequentie bevat de waarde 1 op tijdindex 0. Dus:

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

$$\delta = (1, 0, 0, \dots)$$

De impuls is grafisch voorgesteld in Figuur 14. De impuls doet denken aan de Box scaling function, maar het is niet de bedoeling dat er echt een verband is. Het verschil tussen beiden is ook dat de impuls gedefinieerd is in *discrete* tijd en de Box function op *continue* tijd.



Figuur 14: Impuls op $t = 0$

Omdat deze impuls op tijdindex 0 staat, is deze impuls niet verschoven in de tijd.

Laten we nu de convolutie uitrekenen van een willekeurige filter f op deze impuls. In de uitdrukking $h(n) = \sum_{m=0}^n f(m) g(n-m)$ zal nu in de sommatie de term $g(n-m)$ steeds nul zijn

behalve wanneer $m = n$. Want voor $m = n$ geldt $g(n-m) = g(0) = 1$. Bijgevolg wordt $h(n)$ berekend als $f(n)$. We krijgen voor een impuls dus letterlijk onze filter f terug als output. Daarom noemen we het geheel aan filtercoëfficiënten in f ook wel de *impulse response* van deze filter.

We kunnen deze impuls δ ook vertragen (naar rechts schuiven over de tijd-as). Dit wordt genoteerd met een operator S die staat voor "shift". De shift is per definitie naar rechts en introduceert dus een *delay* op de impuls:

$$S\delta(n) = \delta(n-1) = \begin{cases} 1 & n=1 \\ 0 & n \neq 1 \end{cases}$$

$$S\delta = (0, 1, 0, 0, \dots)$$

De shift kan men in deze notatie bekijken als een operator S waar δ langs rechts mee vermenigvuldigd wordt. We zullen hier een conventie van maken: telkens wanneer we een operator toepassen op een sequentie schrijven we hem links van die sequentie. Het is dus ook mogelijk om op die manier meerdere operatoren achter elkaar te schrijven, waarbij de meest rechtse operator het eerst wordt uitgevoerd op de sequentie.

We kijken nu wat er gebeurt als we de convolutie toepassen van een filter op een vertraagde impuls. Stel dat de filter f de volgende coëfficiënten bevat: $f = (a, b, c, d, \dots)$

Als we de convolutie van f met $S\delta$ opschrijven in vermenigvuldigingsnotatie, krijgen we:

$$\begin{array}{r} \cdots \ 0 \ 0 \ 1 \ 0 \ = S\delta \\ \cdots \ d \ c \ b \ a \ = f \\ \hline \cdots \ 0 \ 0 \ 0 \ a \ 0 \\ \cdots \ 0 \ 0 \ b \ 0 \\ \cdots \ 0 \ c \ 0 \\ \cdots \ d \ 0 \\ \hline \cdots \ d \ c \ b \ a \ 0 \ = h \end{array}$$

We concluderen hieruit dat de output ook een kopie van de filter f is, maar net zoals de impuls met één tijdseenheid vertraagd. Dit is te vergelijken naar eender welke vertragingsgrootte op de impuls. Een vertraging op de databasequentie is dus een vertraging op de outputquentie:

$$h = f * S\delta = Sf$$

We kunnen nog meer onderzoeken: we kunnen de impuls ook hoger of lager maken, dus verticaal schaleren in waarde. Op die manier krijgen we eigenlijk een willekeurige samplewaarde die in een gewone databasequentie kan voorkomen. We schaleren de impuls met factor σ :

$$\sigma\delta = (\sigma, 0, 0, \dots)$$

Men mag σ beschouwen als een operator waar een sequentie langs rechts mee vermenigvuldigd wordt, net zoals de shift-operator S van hierboven. Nu volgt rechtstreeks uit de tweede eigenschap van convolutie dat:

$$h = f * \sigma\delta = \sigma(f * \delta) = \sigma f$$

We ontvangen een kopie van de filter, maar geschaleerd. Een schalering van de dasequentie is dus een schaling van de output.

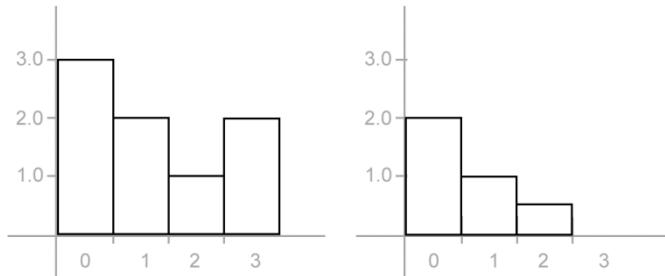
Het vertragen en verticaal schalen van een impuls kan ook gecombineerd worden. Het effect op de output is dan ook zowel een vertraging als een schaling van de filtercoëfficiënten.

Nu komen we tot de betekenis van convolutie. In het algemeen bestaat de dasequentie g uit een reeks getalwaarden, zoals bij het getallen voorbeeld uit de vorige paragraaf. Elk van deze getalwaarden mogen we eigenlijk bekijken als een geschaleerde en verschoven impuls. Met deze zienswijze in het achterhoofd kunnen we laten zien wat convolutie van filter f op dasequentie g eigenlijk betekent.

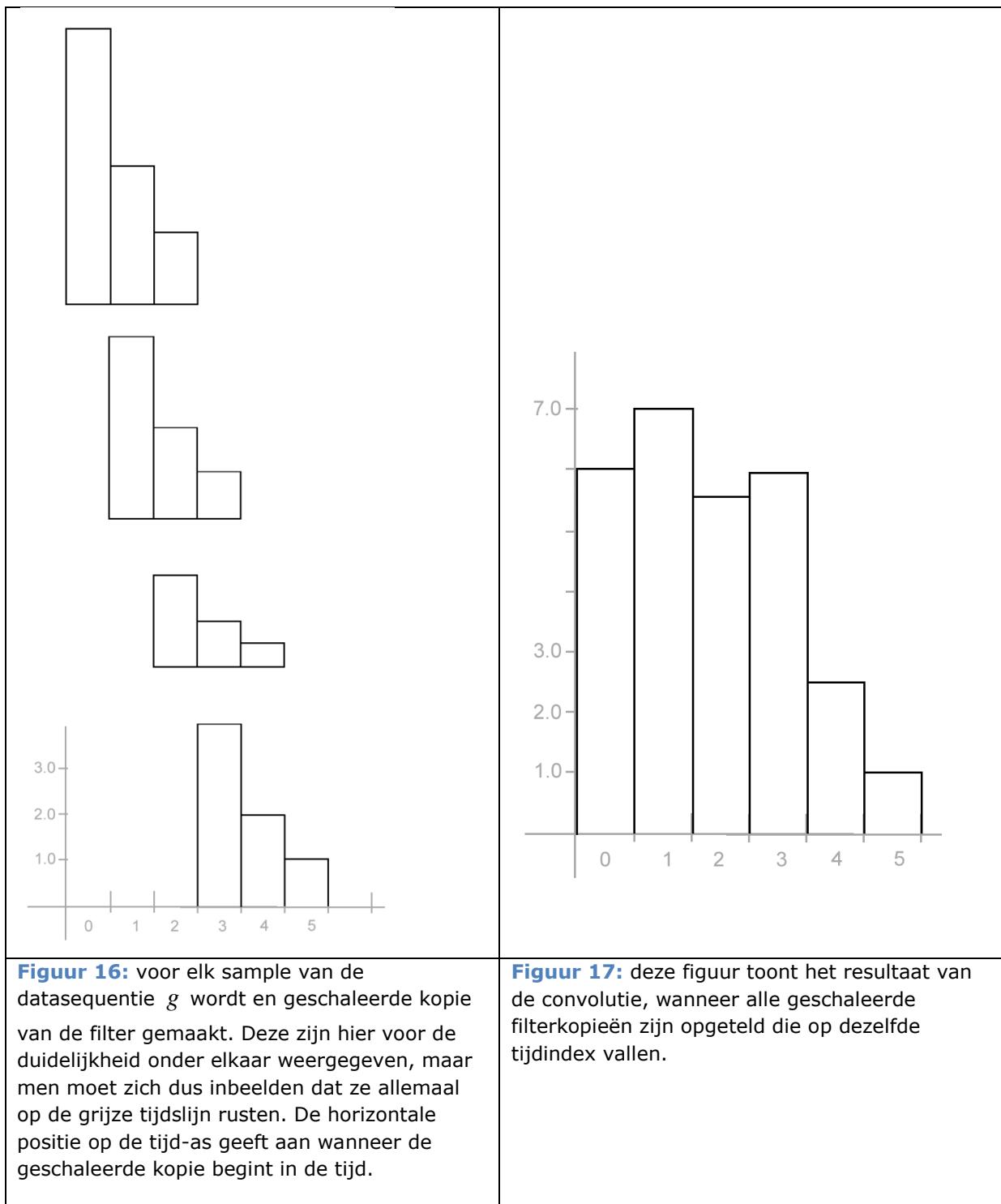
Beschouw in de dasequentie g de samplewaarde op tijdindex t . Noem deze waarde x . We maken nu een kopie van de filtercoëfficiënten uit f . Vervolgens worden de waarden in deze kopie vermenigvuldigd met de waarde x zodat het resultaat xf bekomen wordt. Daarna vertragen we xf met tijd-stap t (verschuiven naar rechts over de tijdsas).

Indien we dit doen voor elk sample uit g en alle bijhorende verschoven xf 's optellen, krijgen we het resultaat van de convolutie.

Dit proces is grafisch voorgesteld in een aantal figuren. In Figuur 15 worden een voorbeeld dasequentie g (links) en een voorbeeld filtersequentie f (rechts) getoond. Figuur 16 toont de verschillende stappen die we hierboven hebben beschreven: voor elk sample in g wordt een verticaal geschaleerde versie van de filter f gemaakt en met de juiste delay op de tijd-as verschoven. In Figuur 17 wordt tenslotte het resultaat van de convolutie getoond dat bekomen wordt door alle geschaleerde filterkopieën op te tellen.



Figuur 15: dasequentie g (links) en filter f



4.5.Tijdscomplexiteit

Zij $h = f * g$. Zij $|f| = F$ en $|g| = N$.

Convolutie vereist FN vermenigvuldigingen. Indien er geen samples in g zitten met waarde 0, dan zal elk sample in de convolutie-som berekend worden met maximaal F optellingen. Dit komt omdat er maximaal F verticaal geschaleerde kopieën van de filter in een tijdindex kunnen aankomen. Omdat de lengte van de convolutie-output $N+F-1$ is, vormt $(N+F-1)F$ een bovengrens op het aantal optellingen dat moet gebeuren per convolutie. Indien we de filters niet tot de input van het algoritme rekenen, gebeurt elke convolutie in lineaire tijd in functie van N .

In totaal is de tijdscomplexiteit van convolutie $h = f * g$ langs boven begrensd door deze uitdrukking:

$$FN + (N+F-1)F$$

4.6. Downsampling

De actie die een filter-bank zal ondernemen is het opsplitsen van een inputsignaal in deelsignalen. In deze tekst staat een filter-bank met één lowpass- en één highpass-filter centraal. Beide filters zullen elk afzonderlijk aanleiding geven tot een deelsignaal dat uit het oorspronkelijke signaal wordt afgeleid door convolutie met de betrokken filter. De twee deelsignalen kunnen apart bewerkt en opgeslagen worden. Omgekeerd kunnen zij terug gecombineerd worden om het oorspronkelijke signaal of een bewerkt signaal te synthetiseren met een omgekeerde lowpass- en highpass-filter.

Maar indien men gewoon convolutie uitvoert zonder extra maatregelen te nemen, zal elk deelsignaal even lang zijn als het oorspronkelijke signaal, waardoor de datahoeveelheid eigenlijk verdubbeld is na het toepassen van de filters. We weten echter dat de feitelijke informatie eigenlijk niet kan toegenomen zijn, dus er is zeker wat redundantie in beide dergelijke deelsignalen. De oplossing om de dubbele datahoeveelheid te vermijden is het downsamplen van elk gevormd deelsignaal: we houden enkel de helft over van de twee berekende deelsignalen. In het Engels wordt dit ook wel *decimation* genoemd. In deze tekst bekijken we enkel een downsampling met factor twee en wordt er steeds gekozen voor het weglaten van de oneven indices, tenzij anders vermeld.

Definitie 5: Downsampling

Het downsamplen met factor 2 van een sequentie x waarbij enkel de samples op even indices worden overgehouden wordt als volgt gedefinieerd:

$$y(n) = g(2n)$$

Hierbij is y de outputsequentie na de downsampling. Dit wordt ook als volgt genoteerd in operatornotatie:

$$y = (\downarrow 2)g$$

Over de lengte van y kunnen we het volgende zeggen:

$$|y| = \left\lceil \frac{|g|}{2} \right\rceil$$

De index n gaat daarom van 0 tot en met $\left\lceil \frac{|g|}{2} \right\rceil - 1$.

Opgelet: voor de meeste databasequenties is deze operatie niet omkeerbaar. Hierdoor kunnen we niet de oorspronkelijke databasequentie terugvinden van vóór de downsampling door een bewerking toe te passen op de gedecimeerde databasequentie. De samples op de oneven indices zijn dan kwijtgespeeld. Voor zogenaamde "band-limited" signalen is downsampling wel omkeerbaar, maar daar gaan we niet dieper op in.

De meeste signalen die we verwerken met onze analysefilters zullen niet band-limited zijn. Mogen we dan eigenlijk wel downsamplen? Ja, we hebben immers twee deelsignalen berekend. De volledige informatie in het oorspronkelijke signaal wordt verdeeld over de twee deelsignalen. Een taak bij het ontwerpen van filters is bijvoorbeeld ervoor zorgen dat de gedecimeerde deelsignalen later terug kunnen gecombineerd worden in het oorspronkelijke signaal. Daartoe moeten we ook eerst een upsampling uitvoeren, zie paragraaf 4.7.

Meer informatie over filters vindt men in (Strang & Nguyen, 1996).

Indien we een sequentie g met 1 tijdseenheid vertragen (shift) en vervolgens downsampling toepassen op het resultaat Sg , zullen we de samples overhouden van g op de oneven indices, met een nul aan het begin veroorzaakt door de shift. Dit komt omdat door het downsamplen van Sg we enkel de samples overhouden op de even indices van Sg , maar afgezien van de nul aan het begin zijn dit precies de samples op de oneven indices van g . We besluiten hieruit dat downsampling geen tijdsonafhankelijke operator is omdat het resultaat heel verschillend is indien we de samples van een sequentie vertragen.

Nu we een beeld hebben van de basiswerking van convolutie, kunnen we een uitbreiding beschouwen. Stel dat we na het uitvoeren van convolutie een "downsampling" willen doen op het resultaat.

In plaats van ná de convolutie, kan dit downsamplen ook tijdens de convolutie (maar niet ervóór). We definiëren deze nieuwe vorm van convolutie als volgt:

$$h' = (\downarrow 2)(g * f)$$

We kunnen nu afleiden hoe de elementen van h' moeten berekend worden door de definitie van de standaard convolutie toe te passen:

$$\begin{aligned} h'(n) &= (g * f)(2n) \\ &= \sum_{m=0}^{2n} f(m)g(2n-m) \end{aligned} \tag{4.2}$$

Er geldt ook:

$$|h'| = \left\lceil \frac{|f * g|}{2} \right\rceil = \left\lceil \frac{|f| + |g| - 1}{2} \right\rceil$$

De index n in (4.2) loopt dus van 0 tot en met

$$|h|-1 = \left\lceil \frac{|f|+|g|-1}{2} \right\rceil - 1$$

Merk op dat de stapgrootte in datasequentie g nu tweemaal zo groot is. Als we dit uitwerken op ons voorbeeld uit 4.1 (met sequenties van lengte 4) dan krijgen we:

$$n=0 \quad h'(0) = f(0)g(0)$$

$$n=1 \quad h'(1) = f(0)g(2) + f(1)g(1) + f(2)g(0)$$

$$n=2 \quad h'(2) = f(0)g(4) + f(1)g(3) + f(2)g(2) + f(3)g(1) + f(4)g(0)$$

$$\begin{aligned} n=3 \quad & h'(3) \\ &= f(0)g(6) + f(1)g(5) + f(2)g(4) + f(3)g(3) + f(4)g(2) + f(5)g(1) + f(6)g(0) \\ &= f(3)g(3) \end{aligned}$$

Voor $n > 3$ krijgen we alleen nog maar nulwaarden uit de convolutie. Merk op dat het resultaat met deze nieuwe convolutie-vorm een subset oplevert van het resultaat uit de gewone convolutie. Dit is goed merkbaar als we dit nieuwe resultaat ook in tabel-vorm oopschrijven:

$f(0)g(0)$	$f(0)g(2)$		
$f(1)g(1)$		$f(1)g(3)$	
$f(2)g(0)$		$f(2)g(2)$	
		$f(3)g(1)$	$f(3)g(3)$
$h'(0)=h(0)$	$h'(1)=h(2)$	$h'(2)=h(4)$	$h'(3)=h(6)$

In (Strang & Nguyen, Wavelets and Filter Banks, 1996) wordt geschreven hoe deze downsampling in verband staat met speciale filters, maar dat zou ons te ver voeren. In deze tekst is (4.2) eerder een theoretische vorm van convolutie waar downsampling ingebakken zit. We zullen dan in een latere wiskundige afleiding deze vorm kunnen herkennen, maar in een praktische implementatie kan men natuurlijk gewoon eerst een datasequentie convolueren met een filter en pas daarna downsamplen, zonder letterlijk (4.2) te implementeren.

Omdat convolutie commutatief is, moet het volgende gelden:

$$h'(n) = \sum_{m=0}^{2n} f(m)g(2n-m) = \sum_{m=0}^{2n} g(m)f(2n-m)$$

Laten we dit voor de zekerheid testen, door de meest rechtse uitdrukking in de vorige vergelijking uit te werken voor f en g , beiden met lengte 4:

$$n=0 \quad h'(0) = g(0)f(0)$$

$$n=1 \quad h'(1) = g(0)f(2) + g(1)f(1) + g(2)f(0)$$

$$n=2 \quad h'(2) = g(0)f(4) + g(1)f(3) + g(2)f(2) + g(3)f(1) + g(4)f(0)$$

$$n=3 \quad h'(3)$$

$$\begin{aligned} &= g(0)f(6) + g(1)f(5) + g(2)f(4) + g(3)f(3) + g(4)f(2) + g(5)f(1) + g(6)f(0) \\ &= g(3)f(3) \end{aligned}$$

...

Men kan verifiëren dat dit precies overeenkomt met de eerdere uitwerking van hierboven.

4.7. Upsampling

Een andere belangrijke processing-stap die we later zullen nodig hebben is upsampling. Upsampling van een datasequentie g in deze tekst is het tussenvoegen van 0-waarden *tussen* de oorspronkelijke waarden van g . We duiden de upsampling-operatie aan met $(\uparrow 2)$. Hieronder staat de formele definitie van upsampling:

Definitie 6: upsampling
$y(n) = \begin{cases} g(n/2) & \text{even}(n) \\ 0 & \text{anders} \end{cases}$
Dit wordt ook als volgt genoteerd met operatornotatie:
$y = (\uparrow 2)g$
Voor de output y geldt:
$ y = 2 \cdot g - 1$

We definiëren nu convolutie waarbij de datasequentie g eerst ge-upscaled wordt:

$$h(n) = \sum_m^{\lfloor n/2 \rfloor} g(m)f(n-2m) \quad (4.3)$$

Er geldt $|h| = 2 \cdot |g| - 1 + |f| - 1 = 2 \cdot |g| + |f| - 2$. De index n gaat dus van 0 tot en met $|h| - 1 = 2 \cdot |g| + |f| - 3$.

Veronderstel $f = (a, b, c)$ en $g = (x, y, z)$. Laten we g upsampelen tot $g' = (x, 0, y, 0, z)$ en dan de standaard convolutie $f * g'$ via de standaard-convolutie uit (4.1) uitschrijven:

$$\begin{array}{rccccc}
 & z & 0 & y & 0 & x \\
 & & & c & b & a \\
 \hline
 & az & 0 & ay & 0 & ax \\
 & bz & 0 & by & 0 & bx \\
 & cz & 0 & cy & 0 & cx \\
 \hline
 & cz & bz & az+cy & by & ay+cx & bx & ax
 \end{array}$$

Nu gaan we upsampelen en convolutie "tegelijk" doen via vergelijking (4.3):

$$\begin{aligned}
 h(0) &= g(0)f(0) = ax \\
 h(1) &= g(0)f(1) = bx \\
 h(2) &= g(0)f(2) + g(1)f(0) = cx + ay \\
 h(3) &= g(0)f(3) + g(1)f(1) = g(1)f(1) = by \\
 h(4) &= g(0)f(4) + g(1)f(2) + g(2)f(0) \\
 &= g(1)f(2) + g(2)f(0) = cy + az \\
 h(5) &= g(0)f(5) + g(1)f(3) + g(2)f(1) \\
 &= g(2)f(1) = bz \\
 h(6) &= g(0)f(6) + g(1)f(4) + g(2)f(2) + g(3)f(0) \\
 &= g(2)f(2) = cz
 \end{aligned}$$

We zien dat beide outputs hetzelfde zijn.

5. De snelle wavelet transformatie

In dit hoofdstuk wordt vanuit de wiskunde van wavelets een efficiënt algoritme afgeleid om analyse en synthese uit te voeren met wavelets.

Deze tekst is gebaseerd op (Burrus, Gopinath, & Guo, 1998), (Mallat, 1989) en (Strang & Nguyen, Wavelets and Filter Banks, 1996). Deze drie werken zijn niet echt gedetailleerd wat deze afleiding betreft en daarom werden zij samengevoegd tot onderstaande gedetailleerde en hopelijk meer leesbare uiteenzetting.

5.1. Analyse

In deze paragraaf zullen we zien dat het berekenen van de wavelet decompositie van een discrete signaalfunctie neerkomt op convolutie met downsampling. Dit maakt dus een erg efficiënte implementatie mogelijk op de computer.

In het begin werken we nog niet op algemene resolutie-niveaus j , maar veronderstellen we een signaal $f_1(t) \in V_1$ (resolutie niveau 1). $f_1(t)$ kan voorgesteld worden als een lineaire combinatie van de basisvectoren $\varphi_{1,k}(t) = \sqrt{2}\varphi(2t-k)$ uit V_1 .

Zoals hierboven reeds gezien kan V_1 geschreven worden als $V_1 = V_0 \oplus W_0$. Dus $f_1(t)$ is eigenlijk ook een lineaire combinatie van de basisvectoren uit V_0 en W_0 . Deze basisvectoren zijn $\varphi_{0,k}(t) = \varphi(t-k)$ voor V_0 en $\psi_{0,k}(t) = \psi(t-k)$ voor W_0 . Dus:

$$\begin{aligned} f_1(t) &= \sum a_{1,k} \varphi_{1,k}(t) = \sum a_{0,k} \varphi_{0,k}(t) + \sum b_{0,k} \psi_{0,k}(t) \\ &= \sum a_{0,k} \varphi(t-k) + \sum b_{0,k} \psi(t-k) \end{aligned}$$

De coëfficiënten $a_{0,k}$ en $b_{0,k}$ horen conceptueel samen omdat je op elke tijdpositie k bij de gebruikte scaling function (met coëfficiënt $a_{0,k}$) wat detail in de vorm van een wavelet (met coëfficiënt $b_{0,k}$) moet voegen om het oorspronkelijke signaal uit V_1 op die tijdpositie exact terug te krijgen. Indien al het detail van $f_1(t)$ kan voorgesteld worden met enkel de scaling function, dan is de wavelet-coëfficiënt $b_{0,k}$ gewoon 0.

Zoals reeds benadrukt in voorgaande hoofdstukken, veronderstellen we dat de basis van V_1 orthonormaal is en dus de basis van $V_0 \oplus W_0$ ook.

De eis voor orthonormaliteit (i.p.v. orthogonaliteit) zorgt ervoor dat de formules hieronder eenvoudiger hanteerbaar zijn omdat de ontbinding van een functie in een basis dan eenvoudig kan gebeuren via het inner product van die functie met de basisvectoren.

We berekenen een verandering van basis: gegeven de coëfficiënten $a_{1,k}$ van $f_1(t)$ in ruimte V_1 willen we de coëfficiënten $a_{0,k}$ en $b_{0,k}$ in de basis van $V_0 \oplus W_0$. Dit betekent feitelijk dat we approximatie-coëfficiënten $\{a_{1,k}\}$ van $f_1(t)$ in V_1 kunnen bekijken als een nieuwe set approximatie-coëfficiënten $\{a_{0,k}\}$ op een lager resolutie niveau V_0 en een bijhorende set detail-coëfficiënten $\{b_{0,k}\}$ voor W_0 . Samen zullen deze nieuwe set approximatie-coëfficiënten en detail-

coëfficiënten terug alle fijne details van $f_1(t)$ in V_1 beschrijven, alleen op een andere manier. De nieuwe coëfficiënten in de basis $V_0 \oplus W_0$ zijn dus in zekere zin equivalent aan de eerste set coëfficiënten $\{a_{1,k}\}$.

We gaan de moeder scaling function verschuiven met hoeveelheid k (naar rechts). En vervolgens substitueren we de dilation equation (3.5):

$$\begin{aligned}\varphi(t-k) &= \sum_n h(n) \varphi_{1,n}(t-k) \\ &= \sum_n h(n) \sqrt{2} \varphi(2(t-k)-n) \\ &= \sum_n h(n) \sqrt{2} \varphi(2t-2k-n)\end{aligned}$$

We hebben hierboven dus de oorspronkelijke shiftparameter k uit de dilation equation even hernoemd naar n om verwarring te vermijden. Wanneer we $l = n + 2k$ stellen krijgen we:

$$\begin{aligned}\varphi(t-k) &= \sum_l h(l-2k) \sqrt{2} \varphi(2t-l) \\ &= \sum_l h(l-2k) \varphi_{1,l}(t)\end{aligned}\tag{5.1}$$

Vanuit de wavelet equation doen we ongeveer hetzelfde (ook shift met k en variabele-invoering $l = n + 2k$):

$$\begin{aligned}\psi(t-k) &= \sum_n h_1(n) \sqrt{2} \varphi(2(t-k)-n) \\ &= \sum_n h_1(n) \sqrt{2} \varphi(2t-2k-n) \\ &= \sum_l h_1(l-2k) \sqrt{2} \varphi(2t-l) \\ &= \sum_l h_1(l-2k) \varphi_{1,l}(t)\end{aligned}\tag{5.2}$$

Ook hier hebben we de oorspronkelijke shiftparameter k van de wavelet equation hernoemd naar n om verwarring te vermijden.

Nu schrijven we een uitdrukking voor de coëfficiënten van $f_1(t)$ in V_0 uit, en gebruiken we (5.1):

$$\begin{aligned}
a_{0,k} &= \langle f_1(t), \varphi(t-k) \rangle \\
&\equiv \int f_1(t) \varphi(t-k) dt \\
&= \int f_1(t) \sum_l h(l-2k) \varphi_{1,l}(t) dt \\
&= \sum_l h(l-2k) \int f_1(t) \varphi_{1,l}(t) dt \\
&= \sum_l h(l-2k) a_{1,l}
\end{aligned}$$

De uitwerking voor de wavelets is gelijkaardig, gebruik makende van (5.2):

$$\begin{aligned}
b_{0,k} &= \langle f_1(t), \psi(t-k) \rangle \\
&\equiv \int f_1(t) \psi(t-k) dt \\
&= \int f_1(t) \sum_l h_1(l-2k) \varphi_{1,l}(t) dt \\
&= \sum_l h_1(l-2k) \int f_1(t) \varphi_{1,l}(t) dt \\
&= \sum_l h_1(l-2k) \langle f_1(t), \varphi_{1,l}(t) \rangle \\
&= \sum_l h_1(l-2k) a_{1,l}
\end{aligned}$$

Beide resultaten samen:

$$\begin{aligned}
a_{0,k} &= \sum_l h(l-2k) a_{1,l} \\
b_{0,k} &= \sum_l h_1(l-2k) a_{1,l}
\end{aligned} \tag{5.3}$$

Dit is een zeer belangrijke recursieve definitie van de approximatie- en detail-coëfficiënten. De vorm van beide vergelijkingen in (5.3) is eigenlijk convolutie. We zullen dit aantonen door een andere notatie te gebruiken:

Noteer $\tilde{h}(n) = h(-n)$ (met $n = 0 \dots N-1$) dan geldt:

$$h(l-2k) = \tilde{h}(N-1+2k-l)$$

$$a_{0,k} = \sum_l \tilde{h}(2k-l) a_{1,l}$$

Meer concreet vertaald naar sequenties, wordt dit een convolutie van datasequentie a_1 met filter \tilde{h} , met daarbij ook een downsampling:

$$a_0(k) = \sum_l a_1(l) \tilde{h}(2k-l) \quad (5.4)$$

Deze vorm van convolutie hebben we tevoren reeds bestudeerd.

Noteer $\tilde{h}_l(n) = h_l(-n)$, dan geldt dit ook:

$$b_0(k) = \sum_l a_1(l) \tilde{h}_l(2k-l) \quad (5.5)$$

Merk op dat we niets zeggen over de start- en eindwaarden van de index l . Er is geprobeerd deze exact te berekenen uit de bovenstaande afleiding, maar het is moeilijk om deze te laten overeenkomen met de vorm van gelijkheid (4.2). Het zou kunnen dat hier meer theorie voor nodig is, maar de auteurs van de geraadpleegde literatuur concluderen gewoon rechtstreeks dat gelijkheden (5.4) en (5.5) lijken op convolutie met downsampling zonder de start- en eindwaarden van index l uit te rekenen.

De besproken verandering van basis is eigenlijk ook algemeen toepasbaar. In het algemeen geldt dat we coëfficiënten $a_{j+1,k}$ in de basis V_{j+1} kunnen omzetten naar coëfficiënten $a_{j,k}$ en $b_{j,k}$ in de basissen voor V_j respectievelijk W_j . Het is dan de bedoeling dat men dit recursief kan blijven herhalen.

Stelling 1 : Wavelet Decompositie

Een functie $f_{j+1}(t) = \sum a_{j+1,l} \varphi_{j+1,l}(t)$ in ruimte $V_{j+1} = V_j \oplus W_j$ heeft verzamelingen coëfficiënten $\{a_{j,k}\}$ en $\{b_{j,k}\}$ in de nieuwe orthonormale basis $\{\varphi_{jk}(t), \psi_{jk}(t)\}$ als volgt:

$$\begin{aligned} a_{j,k} &= \sum_l h(l-2k) a_{j+1,l} = \sum_l \tilde{h}(2k-l) a_{j+1,l} \\ b_{j,k} &= \sum_l h_l(l-2k) a_{j+1,l} = \sum_l \tilde{h}_l(2k-l) a_{j+1,l} \end{aligned} \quad (5.6)$$

Bewijs:

Voor $j=0$, volgt (5.6) uit (5.3). We zullen nu de stelling voor een algemene j afleiden. We doen ook weer een shift van k naar rechts op een scaling function van niveau j , dit schrijven we vervolgens uit via de dilation equation:

$$\begin{aligned}
\varphi_{j,k}(t) &= 2^{j/2} \varphi(2^j t - k) \\
&= 2^{j/2} \sqrt{2} \sum_n h(n) \varphi(2(2^j t - k) - n) \\
&= 2^{(j+1)/2} \sum_n h(n) \varphi(2^{j+1} t - 2k - n) \\
&= \sum_l h(l - 2k) 2^{(j+1)/2} \varphi(2^{j+1} t - l) \\
&= \sum_l h(l - 2k) \varphi_{j+1,l}(t)
\end{aligned}$$

Hierbij is ook de substitutie $l = 2k + n$ toegepast. Wanneer we dit resultaat verder gebruiken krijgen we:

$$\begin{aligned}
\langle f(t), \varphi_{j,k}(t) \rangle &= \int_{-\infty}^{\infty} f(t) \sum_l h(l - 2k) \varphi_{j+1,l}(t) dt \\
&= \sum_l h(l - 2k) \int_{-\infty}^{\infty} f(t) \varphi_{j+1,l}(t) dt \\
&= \sum_l h(l - 2k) \langle f(t), \varphi_{j+1,l}(t) \rangle \\
&= \sum_l h(l - 2k) a_{j+1,l}
\end{aligned}$$

De afleiding voor de wavelet-kant is gelijkaardig. Q.E.D.

5.2. Tijdscomplexiteit Analyse-gedeelte

Een typische wavelet-analyse bevat meestal meerdere iteraties van (5.6) na elkaar.

Het convolutie-algoritme zoals aangegeven door stelling 1 vereist de approximatie-coëfficiënten op een bepaalde startresolutie. Door telkens recursief de convolutie uit te voeren wordt de set approximatie-coëfficiënten opgedeeld in een tweede set approximatie-coëfficiënten op lager niveau en bijhorende detail-coëfficiënten (=wavelet-coëfficiënten). De recursie gaat enkel verder op de approximatie-coëfficiënten. Als we na een p iteraties zouden stoppen zouden we dan slechts 1 set approximatie-coëfficiënten overhouden en meerdere sets detail-coëfficiënten. Om terug te gaan naar de uiteenzetting van 3.4 starten we met de approximatie-coëfficiënten op een bepaald resolutie niveau J . Via de convolutie splitsen we dan telkens de wavelet-coëfficiënten af:

$$V_J = V_{J-p} \oplus W_{J-p} \oplus \cdots \oplus W_{J-1}$$

Laten we het aantal elementen in een verzameling aanduiden met het # -teken. Het aantal coëfficiënten $a_{j,k}$ dat we na een iteratie van de convolutie overhouden is noteren we dan met $\#\{a_{j,k}\}$.

In de meeste praktische situaties is het aantal filtercoëfficiënten veel kleiner dan het aantal approximatie-coëfficiënten waarmee men het analyse-algoritme opstart. Na elke convolutie op de input approximatie-coëfficiënten en de downsampling van dat resultaat geldt daarom:

$$\#\{a_{j,k}\} = \frac{\#\{a_{j+1,l}\} + \#\{\tilde{h}\} - 1}{2} \approx \frac{1}{2} \#\{a_{j+1,l}\}$$

$$\#\{b_{j,k}\} = \frac{\#\{a_{j+1,l}\} + \#\{\tilde{h}_l\} - 1}{2} \approx \frac{1}{2} \#\{a_{j+1,l}\}$$

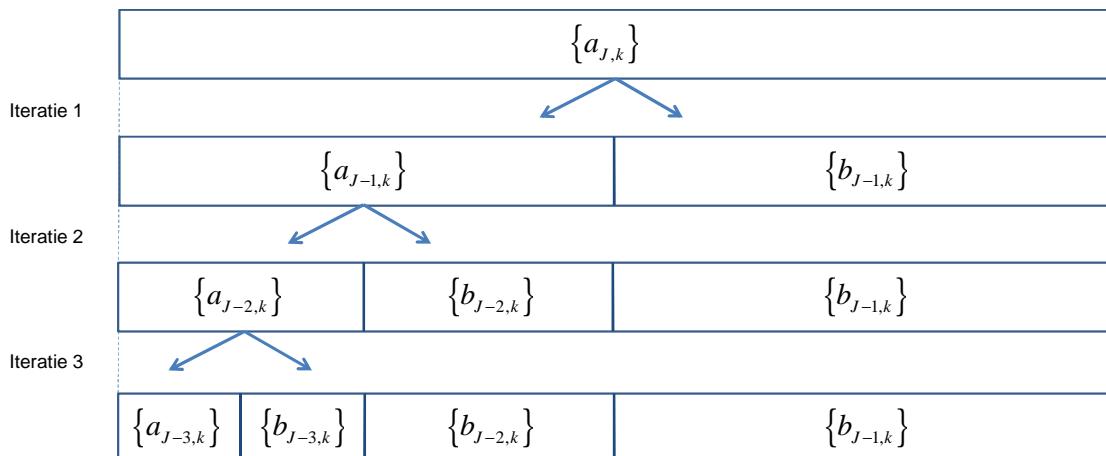
Bijgevolg:

$$\#\{a_{j,k}\} + \#\{b_{j,k}\} \approx \#\{a_{j+1,l}\} \quad (5.7)$$

In paragraaf 5.4 staat aangegeven dat sommige implementaties van convolutie ervoor kunnen zorgen dat er precies evenveel berekende coëfficiënten zijn als samples in de te verwerken signaalsequentie. Maar in onderstaande uitleg zullen we de algemene vorm (5.7) veronderstellen, wat op zich eigenlijk niet zoveel uitmaakt.

Indien we alle coëfficiënten in ons algoritme zouden opslaan in een array van reële getallen, dan zou de plaats die de output van alle berekende coëfficiënten na een aantal iteraties inneemt ongeveer gelijk zijn aan de lengte van de oorspronkelijke input. In Figuur 18 is dit weergegeven voor 3 iteraties. In deze tekst plaatsen we de berekende wavelet-coëfficiënten niet tussen de berekende approximatie-coëfficiënten, maar erachter. Bovendien worden de wavelet-coëfficiënten van verschillende resolutie-niveaus ook niet met elkaar gemengd en netjes achter elkaar geschreven, met het hoogste resolutie-niveau het meest achteraan. Dit is een standaard die in veel teksten en praktische implementaties wordt gehanteerd. In Figuur 18 dient men van boven naar onderen te lezen om de verschillende iteraties van het algoritme uitgevoerd te zien worden. Met een pijltjespaar is telkens aangegeven dat de approximatie-coëfficiënten worden opgesplitst in nieuwe approximatie-coëfficiënten van een lager resolutie-niveau en bijhorende wavelet-coëfficiënten.

Indien het aantal inputcoëfficiënten voor ons algoritme bijvoorbeeld gelijk is aan $N = 2^J$, dan kunnen we $\log_2(2^J) = J$ iteraties uitvoeren. In de allerlaatste stap bekomen we dan 1 approximatie-coëfficiënt en 1 wavelet coëfficiënt.



Figuur 18: 3 iteraties van het analyse-algoritme. Op het einde neemt de array van berekende approximatie en wavelet

coëfficiënten ongeveer dezelfde plaats in als input approximatie-coëfficiënten.

We zullen nu onderzoeken wat de tijdscomplexiteit van het algoritme. Stel dat N de lengte is van de allereerste sequentie van input approximatie-coëfficiënten $\{a_{J,k}\}$ en dat de lengte van beide filters \tilde{h}, \tilde{h}_1 waar we convolutie mee uitvoeren F is. In de eerste analyse-iteratie voeren we convolutie uit met filter \tilde{h} en ook met filter \tilde{h}_1 op. We zullen de filters niet tot de invoer van het algoritme rekenen zodat enkel N echt meetelt in de complexiteit. We voeren tweemaal convolutie uit per analyse-iteratie. Gebruik makende van paragraaf 4.5 is de tijdscomplexiteit van één analyse-iteratie langs boven begrensd door de uitdrukking $T(N)$, afhankelijk van N :

$$\begin{aligned} T(N) &= 2(FN + (N+F-1)F) \\ &= 3FN + 2F(F-1) \end{aligned}$$

Substitueer $a = 3F$ en $b = 2F(F-1)$, er geldt

$$T(N) = aN + b$$

$$T(N/2) = \frac{1}{2}aN + b$$

We zullen de totale tijdscomplexiteit uitrekenen wanneer men een volledige wavelet-analyse wil uitvoeren totdat men nog maar 1 approximatie-coëfficiënt en 1 wavelet coëfficiënt overhoudt. We kunnen maximaal zoveel iteraties uitvoeren als we N kunnen delen door 2, dus $\log(N)$ keer.

Heel precies is het eigenlijk maar $\lfloor \log N \rfloor$ iteraties, maar de eerste is eenvoudiger om te lezen in onderstaande afleiding.

Dus, stel dat we het maximaal aantal iteraties ($\log N$) uitvoeren van het analyse-algoritme. Elke volgende iteratie zal wegens de downsampling telkens een input ontvangen die de helft minder coëfficiënten bevat dan de vorige iteratie. Daarom heeft elke volgende iteratie ook voor de helft minder tijd nodig dan de vorige. De totale tijdscomplexiteit is dan:

$$T(N) + T\left(\frac{N}{2}\right) + T\left(\frac{N}{4}\right) + \dots + T(1)$$

Geschreven als een meetkundige reeks:

$$\begin{aligned} \sum_{i=0}^{\log N} \left((1/2)^i aN + b \right) &= \sum_{i=0}^{\log N} b + aN \sum_{i=0}^{\log N} (1/2)^i \\ &= b \log N + aN \sum_{i=0}^{\log N} (1/2)^i \\ &\leq b \log N + 2aN \\ &= O(N) \end{aligned}$$

Hierbij hebben we op de voorlaatste regel gebruik gemaakt van het feit dat $\sum_{i=0}^{\infty} \frac{1}{2^i} \leq 2$.

Hieruit besluiten we dat de totale tijd van het algoritme, indien de maximale wavelet decompositie wordt uitgevoerd, lineair is in functie van de lengte van het te analyseren signaal. Hetzelfde resultaat geldt ook indien minder iteraties worden uitgevoerd. We besluiten dat dit een zeer efficiënt algoritme is in tijd- en ruimtegebruik.

Tenslotte, nog een korte opmerking. Afhankelijk van de gebruikte scaling function en wavelet zullen de wavelet-coëfficiënten snel of traag naar nul toegaan na een aantal iteraties (Strang, 1989). Hoe beter de vorm van de wavelet de vormen in het signaal beschrijft, hoe vlugger alle detail-informatie uit het signaal wordt gehaald in opeenvolgende iteraties van het algoritme. In dat geval moeten we dus niet het maximaal aantal iteraties uitvoeren! De Haar-wavelet bijvoorbeeld scoort slecht op dit vlak wegens zijn rechthoekige vormen. Voor een signaal met ronde "hellingen" zullen veel iteraties nodig zijn om dit signaal op te delen in wavelet coëfficiënten die horen bij steeds dunnere Haar-wavelets. Men kan intuïtief aanvoelen dat indien de grafiek van de wavelet zelf al ronde hellingen bevat, deze een betere "match" vormen met de ronde hellingen in het signaal.

5.3.Synthese

Wanneer we via het vorige analyse-algoritme eenmaal op een aantal aansluitende resolutieniveaus één set approximatie-coëfficiënten en meerdere sets wavelet coëfficiënten hebben berekend, kunnen we ook terug het oorspronkelijke signaal construeren uit al deze coëfficiënten.

We beginnen de afleiding vanuit deze verandering van basis: gegeven de coëfficiënten van een signaal $f(t)$ in de basis van $V_j \oplus W_j$, bereken de coëfficiënten van $f(t)$ in de basis van V_{j+1} .

De coëfficiënten van $f(t)$ in basis V_{j+1} zijn $c_{j+1,k}$:

$$f(t) = \sum_k c_{j+1,k} \varphi_{j+1,k}(t)$$

De coëfficiënten van $f(t)$ in de basis van $V_j \oplus W_j$ zijn $c_{j,m}$ en $b_{j,m}$:

$$\begin{aligned} f(t) &= \sum_m c_{j,m} \varphi_{j,m}(t) + \sum_m d_{j,m} \psi_{j,m}(t) \\ &= \sum_m c_{j,m} 2^{j/2} \varphi(2^j t - m) + \sum_m d_{j,m} 2^{j/2} \psi(2^j t - m) \end{aligned}$$

Substitueer in vorige vergelijking de dilation equation en de wavelet equation:

$$\begin{aligned} f(t) &= \sum_m c_{j,m} \sum_n h(n) 2^{j/2} \sqrt{2} \varphi(2(2^j t - m) - n) + \sum_m d_{j,m} \sum_n h_1(n) 2^{j/2} \sqrt{2} \varphi(2(2^j t - m) - n) \\ &= \sum_m c_{j,m} \sum_n h(n) 2^{(j+1)/2} \varphi(2^{j+1} t - 2m - n) + \sum_m d_{j,m} \sum_n h_1(n) 2^{(j+1)/2} \varphi(2^{j+1} t - 2m - n) \\ &= \sum_m c_{j,m} \sum_n h(n) \varphi_{j+1,2m+n}(t) + \sum_m d_{j,m} \sum_n h_1(n) \varphi_{j+1,2m+n}(t) \end{aligned}$$

Voor de overzichtelijkheid zullen we schrijven $f(t) = f_c(t) + f_d(t)$ waarbij

$$f_c(t) = \sum_m c_{j,m} \sum_n h(n) \varphi_{j+1,2m+n}(t)$$

$$f_d(t) = \sum_m d_{j,m} \sum_n h_l(n) \varphi_{j+1,2m+n}(t)$$

Werk nu de inner product van $f(t)$ met $\varphi_{j+1,k}(t)$ uit:

$$\begin{aligned} c_{j+1,k} &\equiv \langle f(t), \varphi_{j+1,k}(t) \rangle \\ &= \langle f_c(t) + f_d(t), \varphi_{j+1,k}(t) \rangle \\ &= \langle f_c(t), \varphi_{j+1,k}(t) \rangle + \langle f_d(t), \varphi_{j+1,k}(t) \rangle \end{aligned}$$

We werken vervolgens $\langle f_c(t), \varphi_{j+1,k}(t) \rangle$ apart uit:

$$\begin{aligned} \langle f_c(t), \varphi_{j+1,k}(t) \rangle &= \int f_c(t) \varphi_{j+1,k}(t) dt \\ &= \sum_m c_{j,m} \sum_n h(n) \int \varphi_{j+1,2m+n}(t) \varphi_{j+1,k}(t) dt \\ &= \sum_m c_{j,m} h(k - 2m) \end{aligned} \tag{5.8}$$

Deze uitwerking is mogelijk omdat de uitdrukking $\int \varphi_{j+1,2m+n}(t) \varphi_{j+1,k}(t) dt$ wegens de orthonormaliteit van de basis V_{j+1} gelijk is aan 1 wanneer $k = 2m + n$ en anders 0. Hierdoor zal de binnenste sommatie slechts één $h(n)$ opleveren per waarde van m (k is constant voor de hele expressie). We substitueren vervolgens $n = k - 2m$.

De uitwerking van $\langle f_d(t), \varphi_{j+1,k}(t) \rangle$ is gelijkaardig:

$$\begin{aligned} \langle f_d(t), \varphi_{j+1,k}(t) \rangle &= \sum_m d_{j,m} \sum_n h_l(n) \int \varphi_{j+1,2m+n}(t) \varphi_{j+1,k}(t) dt \\ &= \sum_m d_{j,m} h_l(k - 2m) \end{aligned} \tag{5.9}$$

Combineer nu beide resultaten:

$$\begin{aligned} c_{j+1,k} &= \langle f_c(t), \varphi_{j+1,k}(t) \rangle + \langle f_d(t), \varphi_{j+1,k}(t) \rangle \\ &= \sum_m c_{j,m} h(k - 2m) + \sum_m d_{j,m} h_l(k - 2m) \end{aligned}$$

We kunnen de coëfficiënt-subscripts natuurlijk ook als indices in een array opschrijven:

$$c_{j+1}(k) = \sum_m c_j(m) h(k-2m) + \sum_m d_j(m) h_l(k-2m) \quad (5.10)$$

In paragraaf 4.7 hebben we gezien hoe men een convolutie kan omschrijven waarbij eerst een upsampling gebeurt van de datasequentie. De twee sommaties in vorige vergelijking voldoen beiden aan deze vorm van convolutie. Ook hier hebben we niet exact de start- en eindwaarden van index m berekend, omdat deze niet netjes overeenkomen met de algemene vorm van convolutie met upsampling van de datasequentie (4.3). De auteurs van de geraadpleegde literatuur concluderen rechtstreeks dat de sommaties in (5.10) lijken op convolutie met upsampling, zonder de exacte grenzen van de index m uit te rekenen.

Één iteratie van de wavelet-synthese neemt twee inputs: de approximatie-coëfficiënten $\{a_{j,k}\}$ en de wavelet coëfficiënten $\{b_{j,k}\}$ van hetzelfde resolutie-niveau j . Het algoritme zal beide sequenties coëfficiënten upsampelen tot $\{a'_{j,k}\}$ en $\{b'_{j,k}\}$. Vervolgens zal convolutie uitgevoerd worden van de filter h met $\{a'_{j,k}\}$ en van de filter h_l met $\{b'_{j,k}\}$. Het resultaat van beide convoluties wordt opgeteld en men bekomt de coëfficiënten van resolutie niveau $j+1$.

Men kan net zoveel iteraties van het synthese-algoritme uitvoeren als men data beschikbaar heeft. Omdat dit synthese-algoritme volgt op het analyse-gedeelte, zal het als input twee arrays van reële getallen binnenkrijgen die de output vormden van het analyse-gedeelte. De gebruiker kan natuurlijk wel eerst naar wens de output van het analyse-gedeelte wat manipuleren, zoals bijvoorbeeld een noise-reduction waardoor wavelet-coëfficiënten van fijnere details kleiner gemaakt worden of volledig verwijderd worden.

Per iteratie van het synthese-algoritme wordt wegens upsampling, beide convoluties en de sommatie enkel lineaire tijd gebruikt in functie van de lengten van de twee input coëfficiënt-sequenties.

Waar de downsampling bij het analyse-algoritme ervoor zorgde dat de datastream naar de volgende iteratie toe halveerde, zal de upsampling bij het synthese-algoritme de datastream naar de volgende iteratie toe verdubbelen.

Beschouw het voorbeeld van Figuur 18. De eerste iteratie van het synthese-algoritme zal coëfficiënten $\{a_{J-3,k}\}$ en $\{b_{J-3,k}\}$ samenvoegen tot de coëfficiënten $\{a_{J-2,k}\}$. De tweede iteratie zal de coëfficiënten $\{a_{J-2,k}\}$ en $\{b_{J-2,k}\}$ samenvoegen tot de coëfficiënten $\{a_{J-1,k}\}$, enzovoort.

5.4. Implementatie aspecten

In paragraaf 4.1 hadden we afgesproken dat indien er door indices verwezen wordt naar samples buiten het bereik van de filter- en datasequentie, we hiervoor nullen zouden gebruiken. Dit wordt in de literatuur ook *zero-padding* genoemd omdat de filter- en datasequenties denkbeeldig worden uitgebreid met nullen.

Indien men naar samples buiten het originele bereik van de *filter* verwijst zullen deze zeer vaak gelijk gesteld worden aan nul.

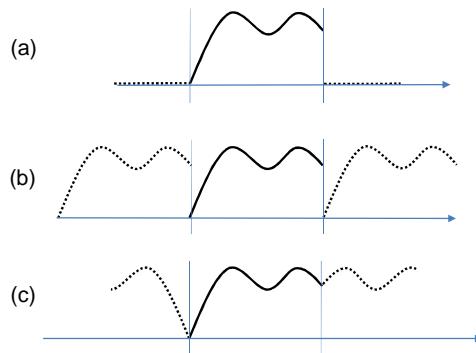
Stel even dat zero-padding gebruikt wordt voor de *datasequentie*. Het is goed mogelijk dat het oorspronkelijke signaal aan zijn randen geen nullen had, waardoor het uitgebreide signaal met zero-padding eigenlijk aan het begin een "sprong" maakt bij de overgang van nullen naar niet-nullen. Ook aan het einde, bij de overgang van niet-nullen naar nullen maakt het signaal dan een sprong. Door deze sprongen zijn de wavelet-coëfficiënten meestal groter om deze abrupte

overgangen te modelleren. Dit kan onwenselijk zijn omdat hierdoor méér wavelet-coëfficiënten van grotere amplitude ontstaan in de ontbinding.

Voor de *databeeld* zijn er ook andere opties die in de literatuur vermeld worden. Zero-padding is slechts één van meerdere zogenaamde *signal-extension* technieken. Andere extension technieken zijn:

- 1) Extension by periodicity (wraparound)
- 2) Extension by reflection (symmetric extension)

In het programma Matlab R2006b (release 7.3.0.267) kan de gebruiker bijvoorbeeld uit de drie genoemde vormen kiezen bij het berekenen van een wavelet decompositie. Deze worden grafisch voorgesteld in Figuur 19. Het aantal toegevoegde samples aan de voor- en achterkant van de databeeld dragen beiden ongeveer de lengte van de filtersequentie. In alle gevallen moeten we dus eigenlijk maar een paar samples buiten het oorspronkelijke bereik van de databeeld bijmaken.



Figuur 19: signal extension technieken; (a) zero-padding, (b) periodic, (c) symmetric

Bij de wraparound doen we alsof de databeeld periodisch is. Stel dat sequentie g een lengte n heeft. Door het toepassen van de periodische extensie geldt:

$$g(i) = g(i \bmod n)$$

Indien we bij het uitrekenen van de convolutie dan een sample $g(n+3)$ nodig hebben, zullen we daarvoor sample $g(3)$ substitueren.

De periodische extensie kan ervoor zorgen dat het aantal approximatie-coëfficiënten en wavelet-coëfficiënten van een wavelet decompositie *precies* gelijk is aan het aantal samples in de oorspronkelijke databeeld. Het geheugenverbruik is dan perfect lineair in functie van dit aantal. In contrast hiermee maken andere extensie-technieken extra samples aan wanneer de filter de randen van de databeeld overschrijdt.

Maar, zoals men kan zien op Figuur 19 heeft de periodische extension techniek ook last van sprongen als de laatste samples van de databeeld niet goed overeenkomen met de eerste samples. De symmetric extension geeft meestal de kleinste hoeveelheid wavelet coëfficiënten aan de randen.

Theorie over extensions kan men vinden in (Strang & Nguyen, Wavelets and Filter Banks, 1996). Een intuïtieve uitleg en grafische voorbeelden kan men ook vinden in (Misiti, Misiti, Oppenheim, & Poggi, 2008).

6. Nabespreking Wavelet Transformatie

Veel van de theorie van wavelets is gebaseerd op eigenschappen in het frequentie-domein. De Fourier transformatie kan gebruikt worden om een datasequentie om te zetten naar zijn voorstelling in het frequentie-domein. Eerst zullen we intuïtief uitleggen wat de Fourier transformatie inhoudt. Daarna vertellen we iets over de interpretatie van convolutie in het frequentie-domein. We sluiten dit hoofdstuk af met het beschouwen van een algemeen filter bank systeem dat gebruikt wordt om de snelle wavelet transformatie mee te implementeren.

6.1. Fourier transformatie

Deze paragraaf is gebaseerd op het werk (Loy, 2007).

Beschouw de volgende definitie:

Definitie: Periodische (Periodieke) functie

Functie $f(t)$ is periodisch met periode $\tau > 0$ als en slechts als:

Voor elke welk punt t in de tijd de waarde $f(t + \tau)$ gelijk is aan $f(t)$. Dus er moet gelden

$$f(t) = f(t + \tau) \quad (-\infty < t < \infty)$$

Een voorbeeld van een periodische functie is de gewone sinusfunctie. Deze heeft periode 2π . Maar de vorm van de curve binnen 1 periode is eigenlijk op geen enkele manier beperkt. Op de grens tussen twee perioden mogen zelfs sprongen voorkomen in de functiewaarden waardoor de functie dus niet continu is.

De Fransman Joseph Fourier heeft het volgende ontdekt:

Eender welk periodisch signaal, onafhankelijk van zijn ogenaanzicht complexiteit, kan beschreven worden als de som van sinusoiden met bepaalde frequenties, amplitudes en fazen.

Definitie: Sinusoïde

We definiëren voor de zekerheid nog het begrip sinusoïde. De functie

$$y(t) = A \cdot \sin(\omega t + \theta)$$

is een sinusoïde met amplitude A , frequentie ω en fase θ .

De functiewaarden $y(t)$ liggen in het interval $[-A, A]$. De frequentie bepaalt hoe vaak de "golven" van de sinusfunctie op en neer gaan. De fase tenslotte is de het punt op de tijd-as waar de eerste lobbe van de sinusfunctie vertrekt, dus een soort van horizontale verschuiving over de tijd-as.

Een periodisch signaal heeft dus sinusoïden als bouwstenen, ook wel spectrale componenten genoemd. Door het toepassen van Fourier Analyse zal men een signaal $x(t)$ in het tijdsdomein kunnen interpreteren als een beschrijving van de spectrale componenten (en hun sterktes) die moeten opgeteld worden om $x(t)$ te bekomen. Men bekomt dan een tweede functie $X(\omega)$, het "spectrum" genaamd, in het frequentie-domein. Een spectrum is een specificatie van de sterktes van de sinusoïden. Hierbij komt elke frequentie ω overeen met een sinusoïde met die frequentie. Wanneer men een grafiek zou maken van het spectrum, wordt meestal ω op de X-as uitgezet en de sterktes van de bijhorende sinusoïden op de Y-as.

Fourier Synthese is de omgekeerde richting van Fourier Analyse. Men via Fourier Synthese een signaal $x(t)$ in het tijdsdomein maken door de frequenties te selecteren in een spectrum die een sterkte verschillend van 0 hebben en deze sterktes te vermenigvuldigen met de bijhorende sinusoïde op die frequentie, rekening houdend met de faze van die sinusoïde. Omdat sinusoïden periodisch zijn, zal het gesynthetiseerd signaal ook periodisch zijn.

We herinneren ons van de middelbare school de formule van Euler:

Formule van Euler
$e^{i\theta} = \cos(\theta) + i \sin(\theta)$
Deze formule beschrijft een punt (vector) op de eenheidscirkel in het complexe vlak. De projectie van $e^{i\theta}$ op de reële as levert $x = \cos(\theta)$ op en de projectie op de imaginaire as levert $y = \sin(\theta)$ op.
Men kan deze notatie $e^{i\theta}$ natuurlijk zien als een afkorting van een complex getal en de notatie verder gewoon als "black box" hanteren. Na deze paragraaf zal dat ook op die manier gebeuren.

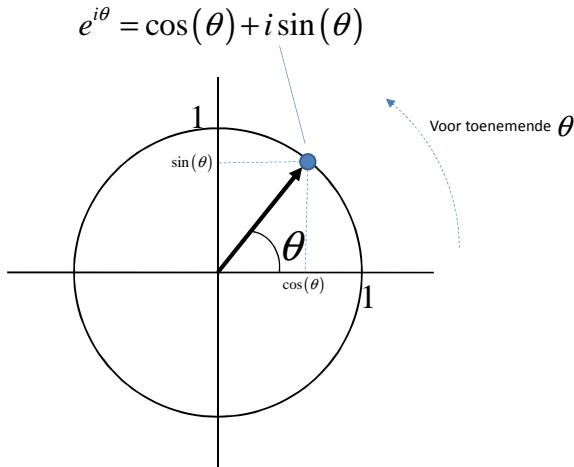
Men noemt $e^{i\theta}$ een "fasor". We kunnen met een fasor elk complex getal z beschrijven in zijn polaire voorstelling:

$$z = r e^{i\theta}$$

Concreter mogen we schrijven $z = |z| e^{i\theta}$. Hierbij is $|z|$ de "modulus" van het complex getal z .

Als de hoek θ groter of kleiner wordt, zal het punt dat het complex getal represeneert ronddraaien op de cirkel in het complexe vlak. In Figuur 20 is getoond hoe het punt zal bewegen indien de hoek groter wordt.

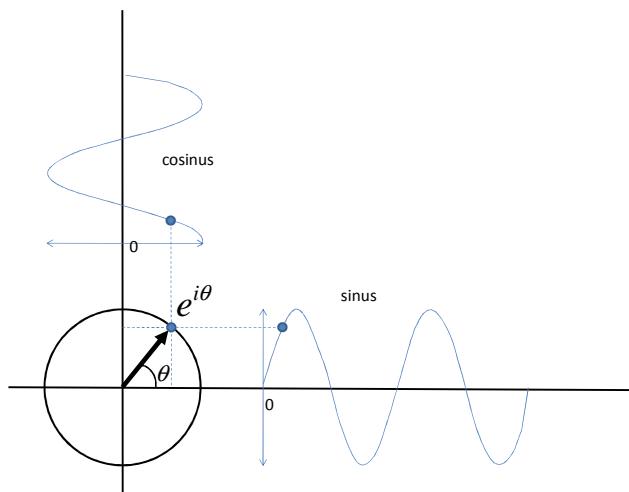
Er zijn positieve-frequentie fasors en negatieve-frequentie fasors. De positieve-frequentie fasor wordt getoond in Figuur 20. Om een fasor in de omgekeerde richting te laten bewegen dient men een minteken te plaatsen voor de hoek θ : $e^{-i\theta} = \cos(\theta) - i \sin(\theta)$. Indien we θ enkel laten groter worden, zal $e^{i\theta}$ een positieve-frequentie fasor zijn en $e^{-i\theta}$ een negatieve-frequentie fasor.



Figuur 20: positieve-frequentie fasor

Heel belangrijk is het volgende: de straal van de cirkel waarop de fasor ronddraait bepaalt de amplitude van de sinusoïden. Hoe groter de straal van de fasor, hoe groter de amplitude van de sinusoïde.

De fasor begint altijd te draaien vanaf θ gelijk aan 0, waarbij de pijl dus op de reële as ligt. Als we projecteren op de imaginaire as (y-as) volgens de richting van de reële as (x-as), dan is de resulterende sinusoïde de gewone standaard sinusfunctie die waarde 0 heeft in de oorsprong. Wanneer we de fasor projecteren op de reële as, volgens de richting van de imaginaire as, dan is de resulterende sinusoïde de gewone standaard cosinus, waarvan de absolute waarde in de oorsprong gelijk is aan 1. Dit projecteren is geïllustreerd in Figuur 21. De standaard sinus en standaard cosinus zijn beiden sinusoïden, maar met een faze-verschil van $\pi/2$.



Figuur 21: projectie levert gewone sinus en cosinus op

We komen terug op de frequentie waarmee een fasor ronddraait. De fasor $e^{i\theta}$ doet het punt eenmaal rond de cirkel draaien wanneer θ gaat van 0 tot 2π . Als de fasor met een constante snelheid ronddraait geldt dat:

$$\theta = \omega t$$

Hierbij is t de tijdsveranderlike. Dan ziet de bewegende fasor er zo uit:

$$e^{i\omega t}$$

Dat is de vorm die we vanaf nu zullen gebruiken. Met elke verschillende frequentie ω komt een bepaalde draaisnelheid overeen.

De Fourier-transformatie \tilde{F} van een continu signaal $g(t)$ wordt nu als volgt gedefinieerd:

$$\tilde{F}(g(t)) = G(\omega) = \int_{-\infty}^{\infty} g(t) e^{-i\omega t} dt$$

In de formule kan men zien dat we voor elke frequentie ω een fasor maken en deze vermenigvuldigen met het signaal. De integraal over de hele tijd-as van dit productsignaal zegt hoe sterk het signaal $g(t)$ overeenkomt met de sinusoïde die bij die frequentie ω hoort. De resulterende continue functie $G(\omega)$ is het continue spectrum van signaal $g(t)$.

In deze tekst zijn we echter geïnteresseerd in discrete databasequenties, waar de waarden slechts op discrete tijdstippen gekend zijn. In digitale signaalverwerking voert men daarom de *discrete-time Fourier transform* in:

$$G(\omega) = \sum_{n=0}^{N-1} g(n) e^{-i\omega n}$$

Hierbij is g een databasequentie van lengte N en tijd wordt voorgesteld met een variabele $n \in \mathbb{N}$. Voor de eenvoud noemt men ω hier ook de frequentie. Wanneer we in de rest van de tekst spreken over de Fourier-transformatie, veronderstellen we de discrete-time versie. Maar omdat de tijdstippen van databasequentie g niet lopen van $-\infty$ tot ∞ , mogen we in principe schrijven:

$$G(\omega) = \sum_{n=0}^{N-1} g(n) e^{-i\omega n}$$

Alle andere samples buiten interval $[0, N-1]$ worden dus nul verondersteld.

We zullen de discrete-time vorm hierna niet explicet gebruiken omdat we enkel Fourier transformaties beschouwen op een theoretisch niveau, met functies i.p.v. sequenties.

6.2. Convolutie is vermenigvuldiging in frequentie-domein

In deze paragraaf zullen we op een intuïtieve manier bespreken wat de betekenis van convolutie is in het frequentiedomein. Deze paragraaf is gebaseerd op het werk (Strang & Nguyen, 1996) en (Hecht, 1998)

Zij $f(t) \in L^2(\mathbb{R})$ en $g(t) \in L^2(\mathbb{R})$ twee functies met $t \in \mathbb{R}$. We definiëren het convolutie-product $h(t) \in L^2(\mathbb{R})$ van beiden als volgt:

$$h(t) = \int_{-\infty}^{\infty} f(u)g(t-u)du \text{ met } t,u \in \mathbb{R}$$

We noteren ook: $h = f * g$, net zoals de convolutie voor sequenties.

Stelling 2: Convolutie-stelling

Zij f en g twee functies uit $L^2(\mathbb{R})$. Zij $h \in L^2(\mathbb{R})$ het convolutie-product van beiden:

$$h = f * g$$

Dan geldt:

$$\begin{aligned} H(\omega) &= \tilde{F}(h) \\ &= \tilde{F}(f * g) \\ &= \tilde{F}(f) \cdot \tilde{F}(g) \\ &= F(\omega) \cdot G(\omega) \end{aligned}$$

In woorden: het spectrum van het convolutie-product van functies f en g is het product van hun beider spectra.

Bewijs:

$$\begin{aligned} H(\omega) &= \int_{-\infty}^{\infty} h(t) e^{-i\omega t} dt \\ &= \int_{-\infty}^{\infty} e^{-i\omega t} \left[\int_{-\infty}^{\infty} f(u) g(t-u) du \right] dt \\ &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} g(t-u) e^{-i\omega t} dt \right] f(u) du \end{aligned}$$

De stap de eerste regel naar de tweede regel gebeurt door te schrijven dat h het convolutie-product is van f en g .

Voor de binnenste integraal op de derde regel voeren we volgende substitutie door:

$$v = t - u$$

Dan geldt (omdat u daar een constante is):

$$dv = v' dt = 1 \cdot dt = dt$$

Vervolgens:

$$\begin{aligned}
 H(\omega) &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} g(v) e^{-i\omega(u+v)} dv \right] f(u) du \\
 &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} g(v) e^{-i\omega v} dv \right] f(u) e^{-i\omega u} du \\
 &= \int_{-\infty}^{\infty} G(\omega) f(u) e^{-i\omega u} du \\
 &= G(\omega) \cdot F(\omega)
 \end{aligned}$$

Q.E.D.

In de vorige paragraaf hebben we ook vermeld dat een complex getal mag geschreven worden als de vermenigvuldiging van zijn modulus met de fasor (die staat onder een bepaalde hoek θ).

Omdat een waarde uit de sequentie F (getransformeerde van sequentie f) een complex getal voorstelt dat afhangt van de frequentie waarop we kijken, zal de hoek van de fasor ook moeten afhangen van de frequentie. Dit drukken we uit voor de drie fasors:

$$\begin{aligned}
 F(\omega) &= |F(\omega)| \cdot e^{i\alpha(\omega)} \\
 G(\omega) &= |G(\omega)| \cdot e^{i\beta(\omega)} \\
 H(\omega) &= |H(\omega)| \cdot e^{i\gamma(\omega)}
 \end{aligned}$$

Laten we met dit in het achterhoofd de convolutie uitschrijven van een filter f met een datasequentie g in het frequentie-domein:

$$\begin{aligned}
 H(\omega) &= |H(\omega)| \cdot e^{i\gamma(\omega)} \\
 &= F(\omega)G(\omega) \\
 &= |F(\omega)| \cdot e^{i\alpha(\omega)} \cdot |G(\omega)| \cdot e^{i\beta(\omega)} \\
 &= |F(\omega)| \cdot |G(\omega)| \cdot e^{i(\alpha(\omega)+\beta(\omega))}
 \end{aligned}$$

Hieruit volgt dat $|H(\omega)| = |F(\omega)||G(\omega)|$ en dat $\gamma(\omega) = \alpha(\omega) + \beta(\omega)$. Deze twee vergelijkingen drukken het "effect" van convolutie met een filter in het tijdsdomein uit in het frequentie-domein. Zoals we kunnen zien in de eerste vergelijking zal de filter de amplitude (sterkte) van de sinusoïde bij frequentie ω manipuleren door zijn eigen amplitude op die frequente ermee te vermenigvuldigen. Indien men dit over alle frequenties heen beschouwt, zal de filter de vorm van het spectrum van de datasequentie manipuleren. Bovendien zal door de tweede vergelijking de faze van de resulterende sinusoïde zijn aangepast t.o.v. de oorspronkelijke sinusoïde van de datasequentie op frequentie ω . Zoals eerder opgemerkt is de faze de hoek van de fasor bij $t = 0$.

Eerder hebben we de waarden in f ook *impuls response* genoemd omdat we bij een impuls als datasequentie letterlijk een kopie van de filter terugkrijgen. Wanneer $g = \delta$, de unit impuls op $t = 0$, dan is de output van convolutie dus $h(n) = f(n)$.

We definiëren dat in het spectrum van een unit impuls in het tijdsdomein alle frequenties de sterkte 1 aannemen. Het spectrum van een unit impuls is dus een horizontale rechte door $y = 1$. Wanneer $g = \delta$, dan geldt dus $G(\omega) \equiv 1$ en ook:

$$\begin{aligned} H(\omega) &= F(\omega)G(\omega) \\ &= F(\omega) \end{aligned}$$

Daarom heet $F(\omega)$ ook wel de *frequency response*.

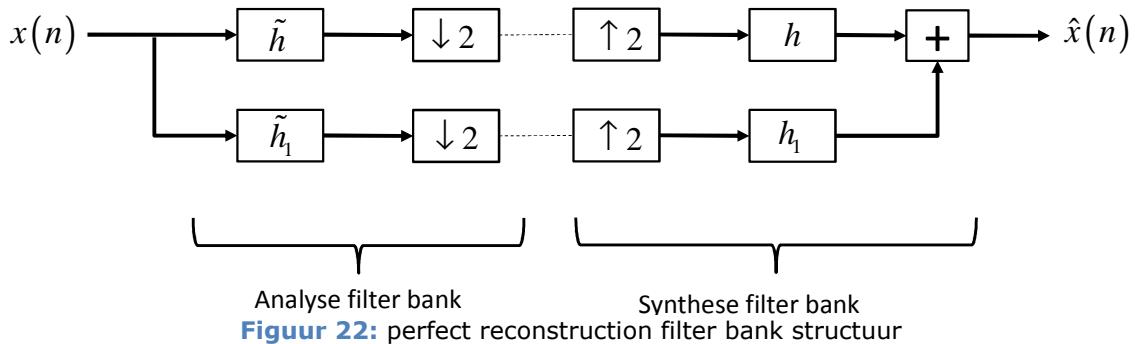
Figuur 24 is een visuele weergave van een highpass en lowpass filter in het frequentie-domein.

6.3. Link naar filter banks

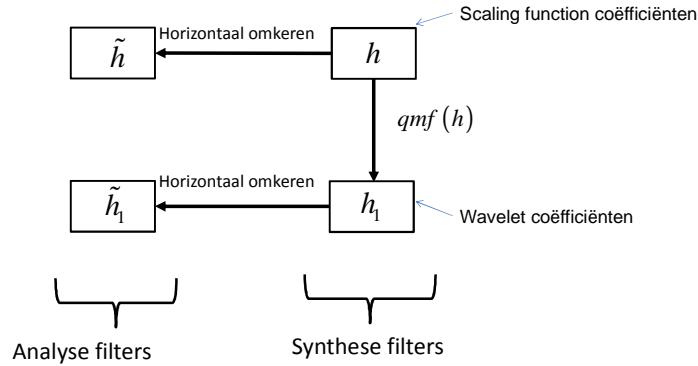
In deze paragraaf leggen we kort het concrete verband tussen wavelets en filter banks. Deze paragraaf is gebaseerd op (Strang & Nguyen, 1996).

Zoals tevoren vermeld, bestaan de analyse filter-bank en de synthese filter-bank die wij gebruiken uit twee filters. Verder willen we dat wanneer we de output van de analyse filter-bank geven als input aan de synthese filter-bank we het oorspronkelijke signaal terugkrijgen. Zo'n paar van filter-banks noemt men *perfect reconstruction filter banks*. In Figuur 22 is het model weergegeven waar we mee werken. Men kan zien op deze figuur dat de filters zijn ingevuld die we in paragrafen 5.1 en 5.3 hebben afgeleid voor wavelets en scaling functions. Merk op dat in Figuur 22 stippenlijntjes getekend zijn tussen het Analyse gedeelte en het Synthese gedeelte. Deze drukken uit dat de gebruiker op die plaats eventueel een eigen algoritme kan uitvoeren waarmee de scaling function en wavelet coëfficiënten bewerkt worden (compressie, ...). Uiteraard, wanneer de coëfficiënten veranderd worden zal niet gelden $x(n) = \hat{x}(n)$. Maar de gebruiker kan natuurlijk op de plaats van de stippenlijntjes ook een verliesloze compressie uitvoeren waarbij de coëfficiënten met grote amplitude met meer bits voorgesteld worden dan de coëfficiënten met een lage amplitude.

De afhankelijkheden tussen alle filters van Figuur 22 is samengevat in Figuur 23. We zien dat de scaling function coëfficiënten h aan de basis liggen van alle filters! De operatie $h_l = qmf(h)$ voert precies gelijkheid (3.10) uit en staat voor *Quadrature Mirror Filter*. Dit is een term uit de signaal verwerkingsliteratuur die gegeven wordt aan bepaalde paren van filters opdat perfect reconstruction mogelijk is.

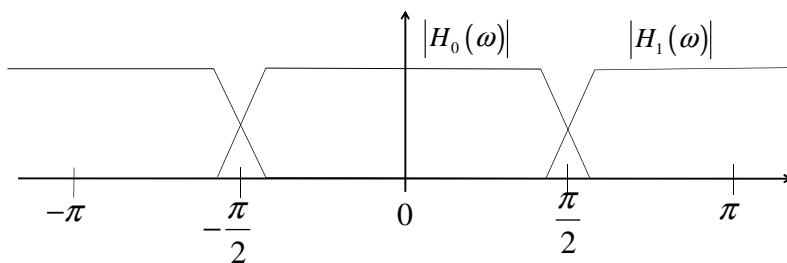


Figuur 22: perfect reconstruction filter bank structuur



Figuur 23: verband tussen de perfect reconstruction filters

De scaling function coëfficiënten worden zodanig ontworpen dat filtersequentie \tilde{h} een lowpass-filter is en \tilde{h}_1 een highpass filter. De algemene vorm van deze filters in het frequentie-domein is getoond in Figuur 24. We zien in deze figuur dat beide filters geen perfecte *bakstenen muurtjes* vormen, maar er zal typisch een overlap zijn tussen beide filters in het frequentie-domein. Hierdoor zal elke filter eigenlijk net iets te veel frequenties doorlaten dan feitelijk tot zijn taak behoort. De synthese-filters h en h_1 moeten hiermee rekening houden en zorgen dat er geen ongewenste vervormingen optreden. We zullen de theorie hiervoor niet bekijken, deze wordt bijvoorbeeld uitgewerkt in (Strang & Nguyen, 1996). Maar men kan zien dat alle filter-eigenschappen zich vertalen in een hele theorie over hoe men scaling functions en wavelets moet ontwerpen.



Figuur 24: highpass en lowpass filter in frequentie domein

7. Image Querying met wavelets

In dit hoofdstuk bespreken we een nuttige toepassing van wavelets in de context van similarity searching: image querying. Dit hoofdstuk is inhoudelijk gebaseerd op het werk (Stollnitz, Derose, & David, 1996), hoewel veel details omtrent de 2D wavelet transformatie zelf werden aangevuld. Alle voorbeelden omtrent wavelet-decomposities zijn zelf gemaakt met het programma Matlab R2006b (release 7.3.0.267). De gebruikte voorbeeldafbeelding (Figuur 38) hoort bij dit product.

We zullen naar de wavelet decompositie/analyse van alle vorige hoofdstukken verwijzen als de één-dimensionale wavelet decompositie. Voordat we aan image querying kunnen beginnen moeten we eerst definiëren hoe men een 2D wavelet decompositie kan uitvoeren van een afbeelding. We veronderstellen in dit hoofdstuk dat er een extension-techniek wordt gebruikt waarbij het aantal berekende coëfficiënten precies gelijk is aan het aantal samples. Daardoor zal de 2D wavelet decompositie van een afbeelding precies even groot zijn als de afbeelding zelf en dit maakt het concept van image querying eenvoudiger om te begrijpen en uit te leggen.

Zoals reeds geïllustreerd in Figuur 18, plaatsen we steeds de berekende wavelet-coëfficiënten achter de berekende approximatie-coëfficiënten en worden de wavelet-coëfficiënten van verschillende resolutie-niveaus ook netjes achter elkaar geschreven, in volgorde van stijgend resolutie-niveau.

In dit hoofdstuk beschouwen we enkel afbeeldingen die bestaan uit bitmap-data: een grid van pixels in een bepaalde color space. Een veel gebruikte color space is bijvoorbeeld RGB: aparte kanalen voor rood, groen en blauw. Elke pixel in de afbeelding bevat in dit systeem één rood-component, één groen-component en één blauw-component. We kunnen elke RGB-afbeelding nu bekijken als drie aparte grids van samples, één grid per kleurenkanaal.

Meer formeel, veronderstel dat een afbeelding N pixels in de hoogte heeft en M pixels in de breedte. Deze afbeelding bestaat uit een C aantal kanalen en per kanaal zijn er $(N \times M)$ samples aanwezig (in grid-vorm). Grids zullen we in deze tekst aanduiden met G . Elke individuele rij of kolom van zo'n grid kan beschouwd worden als een sequentie van samples.

In de context van digitale afbeeldingen wordt de linkerbovenhoek als oorsprong aangeduid, waarbij de Y-as dus naar beneden wijst. In alle voorgaande hoofdstukken hebben we t als veranderlijke gebruikt voor één-dimensionale functies. In dit hoofdstuk moeten we een onderscheid maken tussen horizontale en verticale verschuivingen. Hiervoor zullen we respectievelijk de veranderlijken x en y gebruiken. Verschuivingen op de X-as worden aangeduid met k en verschuivingen op de Y-as met l . Indien we spreken over een sample in een afbeeldings-grid op een bepaalde rij l en kolom k schrijven we $G_{(l,k)}$. Tenslotte, in het tweedimensionale geval kunnen functies ook geschaleerd en verschoven worden:

$$\begin{aligned}\varphi_{j,k}(x) &= 2^{j/2} \varphi(2^j x - k) \\ \varphi_{j,l}(y) &= 2^{j/2} \varphi(2^j y - l)\end{aligned}$$

In dit hoofdstuk zullen we, net zoals bij de één-dimensionale analyse, approximatie-coëfficiënten en wavelet coëfficiënten berekenen, maar dan voor afbeeldingen. We zullen zelden expliciet spreken over "2D coëfficiënten", en meestal gewoon over "coëfficiënten".

Tenslotte nog kort volgende opmerking. Wavelet-analyse op afbeeldingen is populair in de context van zogenaamde *lossy image compression*. Daar worden de approximatie- en wavelet-coëfficiënten op een strategische manier verwijderd zodat de resulterende afbeelding nog steeds zeer goed lijkt op de oorspronkelijke versie, maar minder geheugen in beslag neemt. In (Stollnitz,

Deroose, & David, 1996) wordt uitgelegd hoe dit werkt voor de Box scaling function en Haar wavelet.

7.1. 2D Wavelet Decompositie

We beschrijven de wavelet decompositie van een afbeelding voor de eenvoud slechts voor de samples van één kleurenkanaal omdat de decompositie precies hetzelfde is voor elk kleurenkanaal. We kunnen voor de eenvoud zelfs even vergeten dat er meerdere kleurenkanalen zijn en de afbeelding beschouwen als een eenvoudig ($N \times M$) grid van samples. We weten niet precies op welk resolutie-niveau de afbeelding zich bevindt, maar we zullen dit niveau aanduiden met een onbekende J zoals we ook gedaan hebben op het einde van paragraaf 3.4.1. We doen dus alsof de samples in elke rij en kolom de coëfficiënten zijn van scaling functions uit V_J .

Men zou kunnen overwegen om bij de 2D wavelet decompositie gewoon de samples uit alle rijen of kolommen van een afbeelding achter elkaar te plaatsen om op die manier een lang eendimensionaal signaal te vormen waarop het standaard wavelet-analyse kan toegepast worden. Het nadeel hiervan is dat er slechts in één dimensie van de afbeelding coherentie van samplewaarden kan uitgebuit worden om approximaties en details te definiëren. In een afbeelding zijn er immers twee dimensies en daarom wordt er expliciet voor gekozen om gebruik te maken van zogenaamde 2D scaling functions en wavelets als basisvectoren om de afbeelding mee te beschrijven.

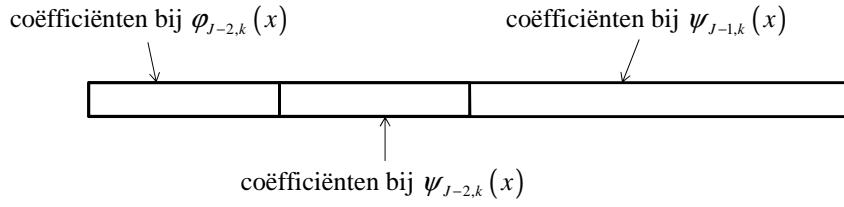
Er zijn twee manieren waarop men een grid van samples kan decomponeren met wavelets: de *standaard decompositie* en de *niet-standaard decompositie*. Bij beide manieren wordt de één-dimensionale decompositie als black-box functionaliteit ingeschakeld. Het kiezen van welke scaling function en bijhorende wavelet men wil gebruiken gebeurt enkel voor deze één-dimensionale decompositie. Er moet dus geen aparte scaling function en wavelet gekozen worden voor de tweedimensionale decompositie als geheel.

2D Standaard decompositie, p iteraties:

- 1) Voer p iteraties van de één-dimensionale wavelet-analyse uit op elke rij van samples in de afbeeldings-grid. Deze operatie levert ons $p+1$ horizontale "zones" in de afbeelding. In de meest linkse zone bevinden zich de approximatie-coëfficiënten. Rechts daarvan bevinden zich in volgende zones wavelet-coëfficiënten. In Figuur 27 is een voorbeeld weergegeven voor $p = 2$, maar op de precieze interpretatie ervan komen we dadelijk terug.
- 2) Beschouw vervolgens de getransformeerde rijen opnieuw als samples uit een grid en pas nu in de verticale richting p iteraties toe van de één-dimensionale wavelet-analyse op elke kolom van de grid. Hierbij is de top van de grid het begin van elke sequentie. Het resultaat hiervan levert links-bovenaan een zone met pure approximatie-coëfficiënten en alle andere zones bevatten wavelet-coëfficiënten op. In Figuur 28 is een voorbeeld weergegeven voor $p = 2$, maar op de precieze interpretatie ervan komen we dadelijk op terug.

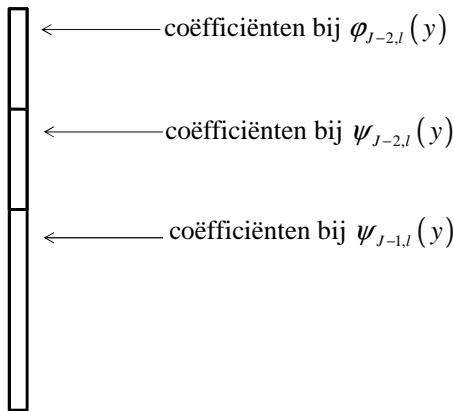
Zoals Figuur 28 aantoon bekomt men na p iteraties van de 2D standaard decompositie $(p+1)^2$ zones van coëfficiënten.

Indien we op één rij uit een afbeeldings-grid twee iteraties toepassen van het één-dimensionale wavelet-analyse algoritme en de coëfficiënten op de typische manier opslaan in array-vorm, krijgen we de structuur getoond in Figuur 25.



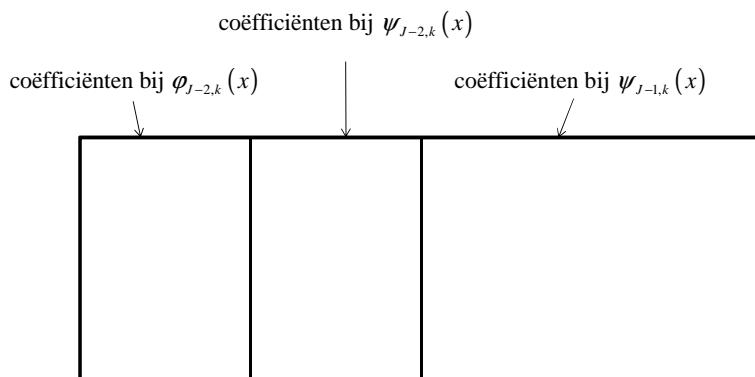
Figuur 25: coëfficiënten na twee iteraties van de één-dimensionale decompositie op één rij uit een grid van samples.

Indien we op één kolom uit een afbeeldings-grid twee iteraties toepassen van het één-dimensionale wavelet-analyse algoritme en de coëfficiënten op de typische manier opslaan, krijgen we de structuur getoond in Figuur 26. Hierbij werd de bovenkant van de afbeeldings-grid beschouwd als het begin van de kolom-datasequenties.



Figuur 26: coëfficiënten na twee iteraties van de één-dimensionale decompositie op één kolom uit een grid van samples.

Indien we in stap 1 van het 2D standaard decompositie algoritme twee iteraties toepassen op elke rij, zien de "zones" eruit zoals getoond in Figuur 27. Dit is een verderzetting van de structuur in Figuur 25. Indien we ook in stap 2) van het 2D standaard decompositie algoritme twee iteraties toepassen op elke kolom zien de zones er vervolgens uit zoals in Figuur 28. We hebben opzettelijk de namen van de basisvectoren weggelaten waar de berekende coëfficiënten bijkoren, want daarover geven we meer uitleg.



Figuur 27: Stap 1 van de 2D standaard-decompositie. De verdeling van de coëfficiënten is getoond na twee

iteraties van de één-dimensionale decompositie op elke rij uit een grid van samples.

Figuur 28: Stap 2 van de 2D standaard-decompositie.

De verdeling van de coëfficiënten is getoond na twee iteraties van de één-dimensionale decompositie op elke kolom.

De standaard decompositie levert na stap 2 scaling function en wavelet coëfficiënten op in een tweedimensionale basis. Deze basis bestaat uit alle mogelijke tensor-producten van de één-dimensionale basisvectoren (=de geschaleerde en verschoven één-dimensionale scaling functions en wavelets).

Definitie: Tensor-product

Deze informatie is gedeeltelijk gebaseerd op (Rowland, 2009).

Zij A en B twee vectorruimten.

Het *tensor-product* van A en B noteren we met $A \otimes B$ en dit is zelf een vectorruimte.

De vectorruimte $A \otimes B$ wordt opgespannen door vectoren (*tensoren*) van de vorm $a_k \otimes b_l$. Bijgevolg is een willekeurige vector in $A \otimes B$ van de vorm:

$$\sum \sigma_{kl} (a_k \otimes b_l)$$

Waarbij de componenten $a_k \in A$, $b_l \in B$ en $\sigma_{kl} \in \mathbb{R}$. Elke tensor is een vector maar niet elke vector is een tensor. We noemen deze som *formeel* omdat elke $a_k \otimes b_l$ één abstract symbol is waarbij niet meteen duidelijk is welke operaties erop zijn toegelaten. In een vectorruimte wil men relaties tussen de vectoren kunnen vaststellen, zoals bijvoorbeeld gelijkheid. In een tensor-product definiëren we volgende *regels* die relaties brengen tussen de formele uitdrukkingen. Zij $a, a_1, a_2 \in A$ en $b, b_1, b_2 \in B$, dan zijn de regels:

- 1) $a_1 \otimes b + a_2 \otimes b = (a_1 + a_2) \otimes b$
- 2) $a \otimes b_1 + a \otimes b_2 = a \otimes (b_1 + b_2)$
- 3) $\sigma(a \otimes b) = (\sigma a) \otimes b = a \otimes (\sigma b)$

Hierbij geldt $\sigma \in \mathbb{R}$. De eerste regel zegt dat twee tensoren mogen opgeteld worden wanneer hun tweede component gelijk is en de tweede regel zegt dat twee tensoren

mogen opgeteld worden wanneer hun eerste component gelijk is. De derde regel zegt hoe men een scalaire vermenigvuldiging kan uitvoeren op een tensor. De scalar mag slechts bij één component geplaatst worden.

Met deze regels is het nu mogelijk om een formele som *uit te werken*. Een voorbeeldje:

$$\begin{aligned}\sigma_1 a_1 \otimes b_1 + \sigma_2 (a_1 + a_2) \otimes b_2 &= a_1 \otimes (\sigma_1 b_1) + (a_1 + a_2) \otimes (\sigma_2 b_2) \\ &= a_1 \otimes (\sigma_1 b_1) + a_1 \otimes (\sigma_2 b_2) + a_2 \otimes (\sigma_2 b_2) \\ &= a_1 \otimes (\sigma_1 b_1 + \sigma_2 b_2) + a_2 \otimes (\sigma_2 b_2)\end{aligned}$$

Hierbij geldt $\sigma_1 a_1 \in A$ en $b_1 \sigma_1, \sigma_2 b_2 \in B$.

We gaan nu over tot het vastleggen van een basis voor het tensor-product, uitgaande van de basissen van A en B . Zij $\{f_k\}$ een basis voor A en zij $\{g_l\}$ een basis voor B . Door gebruik te maken van bovenstaande regels kan men in het algemeen afleiden dat $\{f_k \otimes g_l\}$ voor alle paren (k, l) een basis is van $A \otimes B$. Dit wil zeggen dat een willekeurige vector uit $A \otimes B$ op unieke wijze geschreven kan worden als een lineaire combinatie van deze nieuwe basisvectoren:

$$\forall c \in A \otimes B : c = \sum \sigma_{k,l} f_k \otimes g_l$$

met $\sigma_{k,l} \in \mathbb{R}$. We zullen deze algemene afleiding hier niet maken.

Wij gebruiken in dit hoofdstuk voor A en B concreet de vectorruimte $L^2(\mathbb{R})$. Het tensor-product, functie $(a_k \otimes b_l)(x, y)$, wordt als volgt gedefinieerd:

$$(a_k \otimes b_l)(x, y) = a_k(x) \cdot b_l(y)$$

Met $a_k, b_l \in L^2(\mathbb{R})$. Merk op dat $(a_k \otimes b_l) : \mathbb{R}^2 \rightarrow \mathbb{R}$, en dus een functie is in het vlak.

Opmerking: het symbool " \otimes " betekent niet automatisch een écht product.

Zij nu $\{f_k\}$ en $\{g_l\}$ orthonormale basissen voor A respectievelijk B . We kunnen concreet aantonen dat hierdoor alle tensor-producten $f_k \otimes g_l$ een orthonormale basis vormen voor de 2D functies in $L^2(\mathbb{R}) \otimes L^2(\mathbb{R})$.

Het inner product van twee functies $r : \mathbb{R}^2 \rightarrow \mathbb{R}$ en $s : \mathbb{R}^2 \rightarrow \mathbb{R}$ is een eenvoudige veralgemening van het één-dimensionale inner product als volgt:

$$\langle r, s \rangle = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} r(x, y) \cdot s(x, y) dx dy$$

Stel dat f_i, f_k twee basisvectoren zijn van A en g_j, g_l twee basisvectoren zijn van

B . We tonen de *orthogonaliteit* van de twee tensoren $f_i \otimes g_j$ en $f_k \otimes g_l$ aan:

$$\begin{aligned}
\langle f_i \otimes g_j, f_k \otimes g_l \rangle &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (f_i \otimes g_j)(x, y) \cdot (f_k \otimes g_l)(x, y) dx dy \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_i(x) g_j(y) \cdot f_k(x) g_l(y) dx dy \\
&= \int_{-\infty}^{\infty} f_i(x) f_k(x) dx \cdot \int_{-\infty}^{\infty} g_j(y) g_l(y) dy \\
&= \langle f_i, f_k \rangle \cdot \langle g_j, g_l \rangle \\
&= \delta_{i,k} \cdot \delta_{j,l}
\end{aligned}$$

We tonen aan dat de norm van een tensor $f_i \otimes g_j$ gelijk is aan één:

$$\begin{aligned}
\|f_i \otimes g_j\|^2 &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |(f_i \otimes g_j)(x, y)|^2 dx dy \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |f_i(x) \cdot g_j(y)|^2 dx dy \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |f_i(x)|^2 \cdot |g_j(y)|^2 dx dy \\
&= \int_{-\infty}^{\infty} |f_i(x)|^2 dx \cdot \int_{-\infty}^{\infty} |g_j(y)|^2 dy \\
&= \|f_i\|^2 \cdot \|g_j\|^2 \\
&= \delta_{i,j}
\end{aligned}$$

Specifiek in de context van 2D wavelet decompositie hebben we in een bepaald resolutieniveau j de orthonormale basisvectoren $\{a_{j,k}(x)\}$ voor de X-as en orthonormale basisvectoren $\{b_{j,l}(y)\}$ voor de Y-as.

We zullen deze definitie van een 2D basis met tensor-product specifiek illustreren voor de Box scaling function en Haar wavelet, die we met $\varphi(t)$ respectievelijk $\psi(t)$ zullen aanduiden in onderstaande tekst. We kunnen in het één-dimensionale geval de basis van V_j beschrijven met $\varphi(t)$ en $\psi(t)$. Indien we V_j schrijven als $V_{j-2} \oplus W_{j-2} \oplus W_{j-1}$, zijn de basisvectoren van V_j :

- 1) Verschuivingen van de scaling function: $\varphi_{j-2,k}(t)$
- 2) Verschuivingen van de wavelet: $\psi_{j-2,k}(t)$
- 3) Verschuivingen van wavelets die de helft dunner zijn dan de vorige wavelets: $\psi_{j-1,k}(t)$

We kunnen de basisvector in de oorsprong van elke vectorruimte V_{J-2}, W_{J-2} en W_{J-1} de "kernbasisvector" noemen omdat alle andere basisvectoren van deze kern-basisvector zijn afgeleid door verschuivingen toe te passen. We hebben in bovenstaand voorbeeld dus drie kern-basisvectoren:

- 1) $\varphi_{J-2,0}(t)$ voor V_{J-2}
- 2) $\psi_{J-2,0}(t)$ voor W_{J-2}
- 3) $\psi_{J-1,0}(t)$ voor W_{J-1}

De kern-basisvectoren voor het tensor product $V_J \otimes V_J$ zijn alle productfuncties getoond in Figuur 29. In Figuur 29 duiden de witte blokjes met een "+" positieve waarden aan en de zwarte blokjes met een "-" negatieve waarden. De precieze waarde van de positieve of negatieve blokjes hangt af van hoe dun de twee vermenigvuldigde functies zijn. Door de normalisatie van elke één-dimensionale basisvector zal een dunnere basisvector immers een grotere amplitude hebben dan een bredere basisvector. In Figuur 29 duiden de grijze zones nulwaarden aan. Deze grijze zones ontstaan omdat één van de vermenigvuldigde functies of beiden daar geen support hebben en daar dus nul zijn. Kern-basisvector $\varphi_{J-2,0}(x)\varphi_{J-2,0}(y)$ kunnen we beschouwen als de 2D scaling function. Het is met verschuivingen van deze basisvector dat de ruwe approximatie van een afbeelding wordt opgebouwd. Alle andere 2D kern-basisvectoren geven aanleiding tot soorten 2D wavelets die extra detail kunnen toevoegen aan deze ruwe approximatie. In Figuur 29 werd de vierkante support van 2D scaling function $\varphi_{J-2,0}(x)\varphi_{J-2,0}(y)$ gebruikt als referentie om de support van de 2D wavelets weer te geven. We kunnen visueel op de figuur duidelijk zien dat indien de verticale en horizontale resolutie toeneemt bij de wavelets, hun support ook kleiner wordt. Hierdoor zijn zij in staat om detail toe te voegen aan de ruwe approximatie die beschreven wordt met verschuivingen van $\varphi_{J-2,0}(x)\varphi_{J-2,0}(y)$.

Hierboven hadden we gezegd dat we doen alsof de oorspronkelijke afbeelding afkomstig is van resolutie-niveau J . De 2D kern-basisvector van $V_J \otimes V_J$ is $\varphi_{J,0}(x)\varphi_{J,0}(y)$. Elke sample $G_{(l,k)}$ (op rij l en kolom k) in de afbeeldings-grid G is een coëfficiënt bij een horizontaal en verticaal verschoven $\varphi_{J,0}(x)\varphi_{J,0}(y)$. De hele afbeelding is een 2D signaal in de vectorruimte $V_J \otimes V_J$ als volgt:

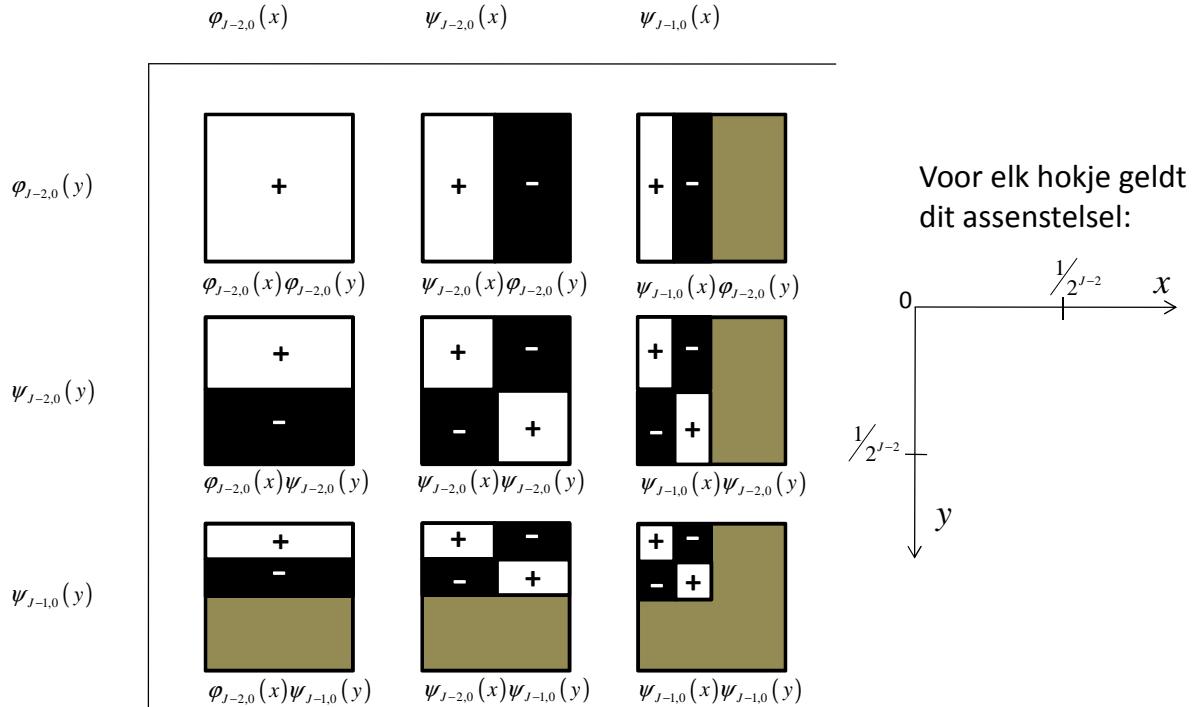
$$\text{afbeelding} = \sum_{l=0}^{N-1} \sum_{k=0}^{M-1} G_{(l,k)} \varphi_{J,k}(x) \varphi_{J,l}(y)$$

Indien de afbeelding grootte $(N \times M)$ heeft zijn er effectief $N \cdot M$ basisvectoren

$\varphi_{J,k}(x)\varphi_{J,l}(y)$ in het vlak (met $0 \leq l < N$ en $0 \leq k < M$) die allemaal zijn afgeleid van kernbasisvector $\varphi_{J,0}(x)\varphi_{J,0}(y)$.

Omdat de afbeelding op de computer bestaat uit pixels kunnen we specifiek voor de Box scaling function de support van elke $\varphi_{J,k}(x)\varphi_{J,l}(y)$ gelijk stellen aan $[0,1] \times [0,1]$. Elke pixel-breedte en pixel-hoogte komt bijgevolg overeen met één eenheid op de X-as respectievelijk op de Y-as. Andere één-dimensionale scaling functions en wavelets hebben een support groter dan één eenheid. Hoe groot de bijhorende 2D basisvectoren voor deze systemen zijn werd niet verder onderzocht in deze tekst.

Doordat we de support van $\varphi_{J,k}(x)\varphi_{J,l}(y)$ gelijk gesteld hebben aan $[0,1] \times [0,1]$, hebben we feitelijk gesteld dat $J=0$. We hebben zelf een referentie resolutie-niveau $J=0$ ingesteld. De 2D wavelet decompositie berekent coëfficiënten in vectorruimten van resolutie $j < 0$, deze coëfficiënten horen bij 2D basisvectoren die een grotere support hebben dan één pixel.



Figuur 29: 2D standaard basis voor $V_J \otimes V_J$, gemaakt uit de Box scaling function en bijhorende Haar wavelet. Deze figuur is gebaseerd op (Stollnitz, Derose, & David, 1996).

In Figuur 30 wordt nogmaals de schikking van de zones herhaald uit Figuur 28 die ontstaat na het uitvoeren van twee iteraties van de 2D standaard wavelet-decompositie. Maar deze keer hebben we ook per zone aangeduid welke 2D kern-basisvector er bij hoort. Elke coëfficiënt in een zone hoort natuurlijk bij precies één 2D basisvector. Elk zulke basisvector in zone Z wordt afgeleid van de 2D kern-basisvector die bij zone Z hoort door horizontale en verticale verschuivingen. In "zone 1" bevinden zich de pure approximatie-coëfficiënten omdat deze horen bij verschuivingen in het vlak van de 2D kern-basisvector $\varphi_{J-2,0}(x)\varphi_{J-2,0}(y)$.

Het is nuttig om even dieper in te gaan op de interpretatie van de coëfficiënten per zone. Elke zone is een soort miniweergave van de hele oorspronkelijke afbeelding, maar niet met alle informatie van de oorspronkelijke afbeelding. Deze interpretatie vloeit op een natuurlijke manier voort uit de interpretatie van coëfficiënten bij de één-dimensionale wavelet-decompositie. Daar werd een eendimensionaal signaal immers geprojecteerd naar een approximatie-ruimte en een aantal wavelet-ruimten. Elke ruimte bevat een deelsignaal, dus een soort miniversie van het oorspronkelijke signaal, maar met ontbrekende informatie. De ontbrekende informatie ligt verspreid over alle ruimten waarop het oorspronkelijke signaal werd geprojecteerd. Beschouw bijvoorbeeld zone 6 van Figuur 30. Hoe breed is deze zone in pixels/eenheden? Veronderstel dat de oorspronkelijke afbeelding grootte $(N \times M)$ heeft. Dan zijn er in de oorspronkelijke afbeelding M kolommen van waarden. Bijgevolg zijn er na de 2D wavelet-

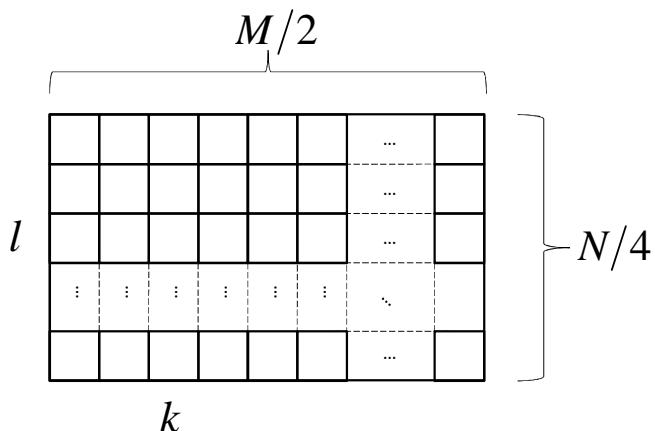
decompositie ook M kolommen van coëfficiënten. Zone 6 ligt helemaal aan de rechterkant en daarom zijn er in zone 6 $M/2$ kolommen met (wavelet) coëfficiënten. Hoe hoog is deze zone in pixels/eenheden? Zone 6 ligt volgens de verticale as in het midden, waardoor er $N/4$ rijen met coëfficiënten zijn. Zone 6 bevat dus $(N \cdot M)/8$ coëfficiënten, intuïtief weergegeven als een grid in Figuur 31. We hebben zojuist opgemerkt dat elke zone, inclusief zone 6, iets zegt over de hele afbeelding. Indien we één coëfficiënt selecteren in zone 6, hoe groot is dan de invloed van deze coëfficiënt op de hele afbeelding? M.a.w. hoe groot is de support in pixels/eenheden van de 2D wavelet waar deze coëfficiënt bij hoort?

We herinneren ons: hoe hoger het resolutie-niveau, hoe kleiner de stapgrootten. Indien we op resolutie-niveau j een stapgrootte s naar rechts maken, komt dit overeen met $s/2^j$ eenheden naar rechts op de tijd-as. Door het invoeren van het referentie resolutie-niveau $J=0$ is de oorspronkelijke afbeelding een signaal in ruimte $V_0 \otimes V_0$. Hierboven hebben we gezegd dat elke pixel grootte $[0,1] \times [0,1]$ heeft. Indien we in een rij pixels van de oorspronkelijke afbeelding van de ene pixel naar zijn rechterbuur toespringen, zal deze sprong overeenkomen met één eenheid naar rechts. De horizontale as van zone 6 bevindt zich op resolutie-niveau -1 . Indien we in een rij van zone 6 springen van een coëfficiënt naar zijn rechterbuur, is deze stapgrootte gelijk aan $1/2^{-1} = 2$ pixels/eenheden in de oorspronkelijke afbeelding. Op een vergelijkbare manier bevindt de verticale as van zone 6 zich op resolutie-niveau -2 . Indien we in een kolom van een coëfficiënt springen naar zijn onderbuur, komt deze sprong overeen met $1/2^{-2} = 4$ pixels/eenheden in de oorspronkelijke afbeelding.

Laten we zone 6 even een lokale oorsprong geven, linksboven. Een hokje (l, k) in dit lokale assenstelsel bevat een 2D wavelet-coëfficiënt. De bijhorende 2D wavelet $\psi_{J-1,k}(x)\psi_{J-2,l}(y)$ is binnen dit lokale assenstelsel horizontaal verschoven met k hokjes naar rechts en verticaal met l hokjes naar onderen. Gebruik makende van bovenstaande uitleg over translatie-grootten, is 2D wavelet $\psi_{J-1,k}(x)\psi_{J-2,l}(y)$ in de oorspronkelijke afbeelding horizontaal verschoven met $2k$ pixels/eenheden naar rechts en verticaal met $4l$ pixels/eenheden naar onderen. Hierboven hebben we gezegd dat elke pixel in de oorspronkelijke afbeelding overeenkomt met een 2D wavelet $\varphi_{J,k}(x)\varphi_{J,l}(y)$ met support $[0,1] \times [0,1]$. Hierdoor is de support van 2D wavelet $\psi_{J-1,k}(x)\psi_{J-2,l}(y)$ uit zone 6 gelijk aan $[0,2] \times [0,4]$ in de oorspronkelijke afbeelding.

<u>Zone 1</u>  $\varphi_{J-2,0}(x)\varphi_{J-2,0}(y)$	<u>Zone 2</u>  $\psi_{J-2,0}(x)\varphi_{J-2,0}(y)$	<u>Zone 3</u>  $\psi_{J-1,0}(x)\varphi_{J-2,0}(y)$
<u>Zone 4</u>  $\varphi_{J-2,0}(x)\psi_{J-2,0}(y)$	<u>Zone 5</u>  $\psi_{J-2,0}(x)\psi_{J-2,0}(y)$	<u>Zone 6</u>  $\psi_{J-1,0}(x)\psi_{J-2,0}(y)$
<u>Zone 7</u>  $\varphi_{J-2,0}(x)\psi_{J-1,0}(y)$	<u>Zone 8</u>  $\psi_{J-2,0}(x)\psi_{J-1,0}(y)$	<u>Zone 9</u>  $\psi_{J-1,0}(x)\psi_{J-1,0}(y)$

Figuur 30: opdeling in zones na twee iteraties van de 2D standaard decompositie. Per zone is de 2D kernbasisvector gevisualiseerd.



Figuur 31: zone 6 uit Figuur 30

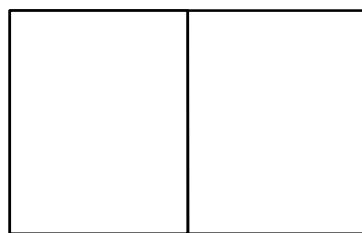
2D Niet-standaard decompositie (één iteratie):

- 1) Pas precies één iteratie toe van het één-dimensionale wavelet-analyse algoritme op elke rij van de afbeeldingsgrid. De linkerkant van de grid is het begin van elke rij-databasequentie.
- 2) Pas precies één iteratie toe van het één-dimensionale wavelet-analyse algoritme op elke kolom. De bovenkant van de grid is de start van elke kolom-databasequentie.

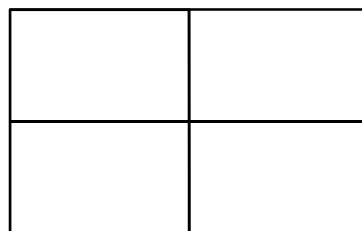
Samen worden stappen 1) en 2) aangeduid als één iteratie. Indien men meerdere iteraties wilt uitvoeren moet men stappen 1) en 2) *recursief* uitvoeren op de zone *linksboven*.

Laten we dit visueel illustreren. In Figuur 32 worden de resulterende zones getoond na stap 1). In Figuur 33 worden de resulterende zones getoond na stap 2), dit is dus het resultaat na één volledige iteratie van de 2D niet-standaard decompositie. Indien we een tweede iteratie uitvoeren, krijgen we Figuur 34 na stap 1) en Figuur 35 na stap 2). Zoals deze figuren aantonen bekomt men na p iteraties $3p+1$ zones van coëfficiënten.

Men kan iteraties van het niet-standaard algoritme blijven herhalen totdat de zone *linksboven* nog slechts uit 1 samplewaarde bestaat. Dit zullen we de *volledige* 2D niet-standaard wavelet decompositie noemen.



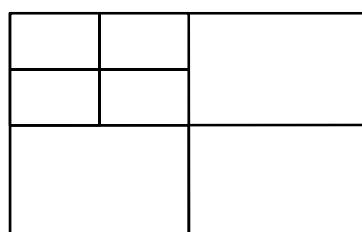
Figuur 32: stap 1 in iteratie 1 van de 2D niet-standaard decompositie.



Figuur 33: stap 2 in iteratie 1 van de 2D niet-standaard decompositie.

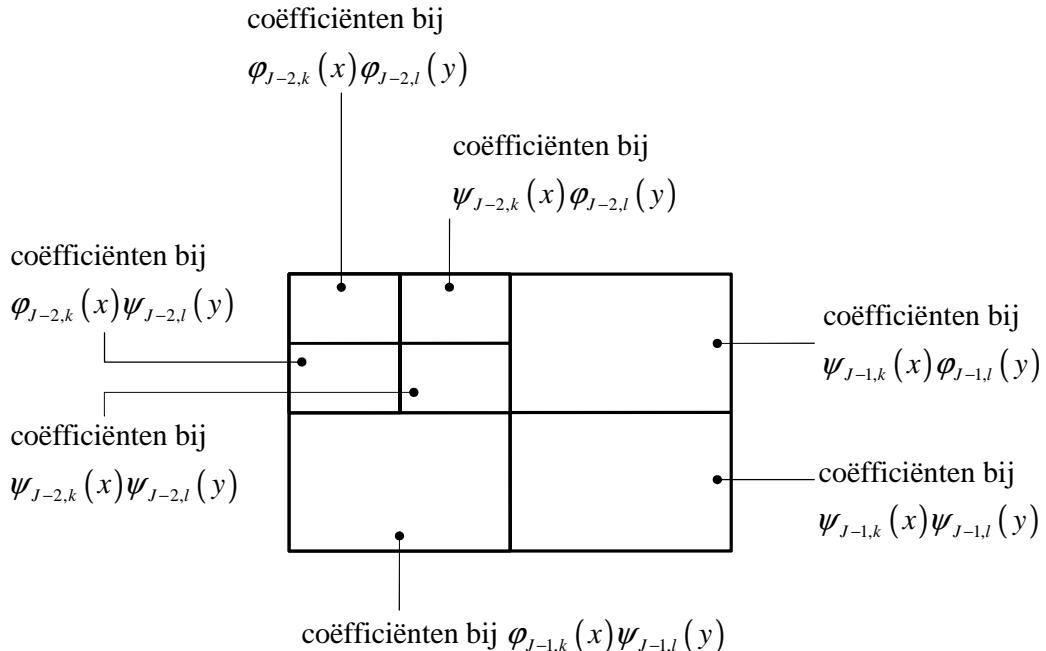


Figuur 34: stap 1 in iteratie 2 van de 2D niet-standaard decompositie.



Figuur 35: stap 2 in iteratie 2 van de 2D niet-standaard decompositie.

In Figuur 36 zijn de formules van de 2D basisvectoren weergegeven. We zullen niet erg diep op de 2D basisvectoren van de niet-standaard decompositie ingaan, omdat eigenschappen van bijvoorbeeld de support van deze 2D basisvectoren op een gelijkaardige manier werken als bij de standaard decompositie.



Figuur 36: de formules van de 2D basisvectoren gevoegd bij de zones uit Figuur 35.

<u>Zone 1</u> $\varphi_{J-2,0}(x)\varphi_{J-2,0}(y)$	<u>Zone 2</u> $\psi_{J-2,0}(x)\varphi_{J-2,0}(y)$	<u>Zone 5</u> $\psi_{J-1,0}(x)\varphi_{J-1,0}(y)$
<u>Zone 3</u> $\varphi_{J-2,0}(x)\psi_{J-2,0}(y)$	<u>Zone 4</u> $\psi_{J-2,0}(x)\psi_{J-2,0}(y)$	
<u>Zone 6</u> $\varphi_{J-1,k}(x)\psi_{J-1,l}(y)$		<u>Zone 7</u> $\psi_{J-1,0}(x)\psi_{J-1,0}(y)$

Figuur 37: opdeling in zones na twee iteraties van de 2D niet-standaard decompositie. Per zone is de 2D kern-basisvector gevisualiseerd.

Wel belangrijk is het volgende: in tegenstelling tot de basis van de standaard-decompositie zijn de 2D basisvectoren bij de niet-standaard decompositie niet de tensor-producten. Voor het geval van de Box scaling function en Haar wavelet, zijn de 2D kern-basisvectoren per zone gevisualiseerd in Figuur 37. 2D kern-basisvector $\varphi_{J-2,0}(x)\varphi_{J-2,0}(y)$ van zone 1 in Figuur 37 is de 2D scaling function waar een ruwe approximatie van de afbeelding mee wordt opgebouwd. In Figuur 37 zien we dat alle andere zones als een soort "schillen" liggen rond zone 1. Deze schil-zones bevatten de coëfficiënten van 2D wavelets. Zones 2 en 5 bevatten coëfficiënten die horen bij wavelets die verticale vormen in een afbeelding waarnemen, waarbij de wavelets van zone 5 kleinere verticale vormen waarnemen dan die van zone 2. Zones 3 en 6 bevatten coëfficiënten die horen bij wavelets die horizontale vormen in een afbeelding waarnemen, waarbij de wavelets van zone 6 kleinere horizontale vormen waarnemen dan die van zone 3. Zones 4 en 7 bevatten coëfficiënten die horen bij wavelets die diagonale vormen waarnemen in een afbeelding, waarbij de wavelets in zone 7 kleinere diagonale vormen waarnemen dan die in zone 4.

Omdat de niet-standaard decompositie belangrijk is in volgende paragraaf zullen we deze eigenschappen bespreken aan de hand van echte afbeeldingen. Bijvoorbeeld, indien we twee iteraties toepassen van de niet-standaard decompositie met de Box scaling function en Haar wavelet op de testafbeelding van Figuur 38, bekomen we de decompositie-afbeelding van Figuur 39. De zwart-wit kleuren van Figuur 39 zijn een visualisatie van de wavelet coëfficiënten: zwart staat voor een lage absolute waarde en wit voor een hoge absolute waarde. Men kan duidelijk zien dat de delen van het gezicht in Figuur 38 gelijkaardige grijstinten bevatten. Hierdoor zullen de wavelet-coëfficiënten voor samples van het gezicht meestal klein zijn, wat aanleiding geeft tot de zwarte gezichten in de wavelet decompositie in Figuur 39. Op plaatsen waar het gezicht eindigt en de kap begint, zullen de wavelet-coëfficiënten eerder hoog zijn omdat in het signaal een "piek" optreedt bij deze overgang. Er is in Figuur 38 bijvoorbeeld een duidelijk contrast tussen de witte kap en het donkere haar en tussen de witte kap en de schaduw op de wang. Dit resulteert in witte randen van de kap rond de zwarte gezichten in Figuur 39, omdat de wavelet-coëfficiënten daar hoog zijn. Merk op dat de kap zelf voortdurend awisselt tussen lichte en donkere streepjes. De resulterende wavelet-coëfficiënten volgen bijgevolg dit patroon.

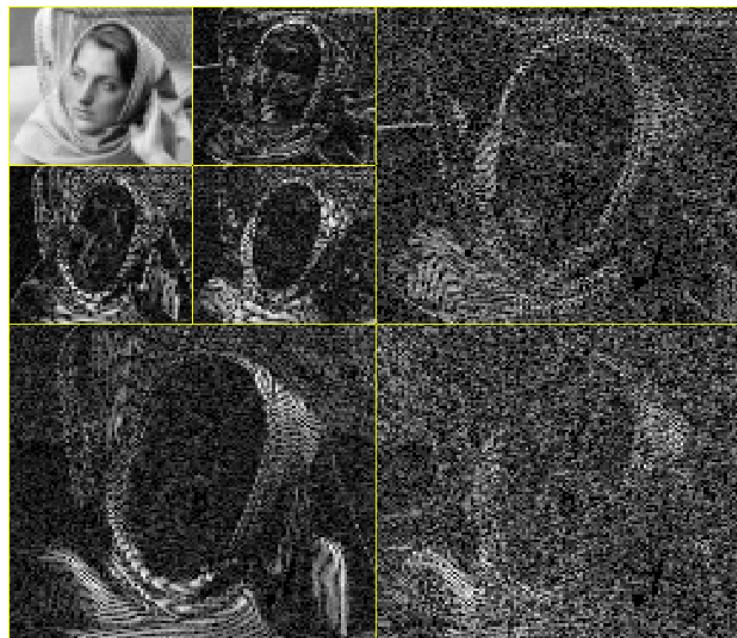
Telkens wanneer men de stappen opnieuw toepast op de zone linksboven, zal deze zone beschouwd worden als een kleinere versie van de oorspronkelijke inputafbeelding. Deze kleinere afbeelding is dan precies één vierde van de oorspronkelijke. Bovendien is deze kleinere versie al gedeeltelijk "beroofd" van hoge frequentie details die zijn achtergebleven in de andere kwadranten in de vorm van wavelet-coëfficiënten. De wavelet-coëfficiënten in het kwadrant links-onderaan geven aan waar horizontale randen of kleurverschillen voorkomen in de afbeelding, omdat deze coëfficiënten berekend worden door de één-dimensionale decompositie toe te passen op de (verticale) kolommen. De wavelet-coëfficiënten die resulteren uit de rijen geven aan waar verticale randen of kleurverschillen voorkomen. Deze coëfficiënten zijn opgeslagen in het kwadrant rechtsboven. Tenslotte zullen de coëfficiënten in het kwadrant rechts-onderaan aangeven waar diagonale randen voorkomen.

Bij de volledige 2D niet-standaard wavelet decompositie blijft in de linkerbovenhoek van de coëfficiënten-grid precies één 2D approximatie-coëfficiënt over, de rest van het grid bevat wavelet coëfficiënten. Deze ene approximatie-coëfficiënt stelt een soort van gemiddelde "kleur" voor van de hele afbeelding.

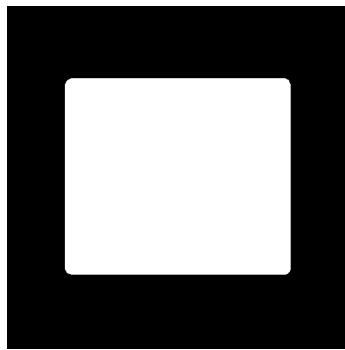
Indien men vijf iteraties van de niet-standaard decompositie met de scaling function en wavelet van de Coiflet-soort (type 1) uitvoert op de testafbeelding uit Figuur 40, bekamt men Figuur 41. In dit voorbeeld wordt het detecteren van de verticale, horizontale en diagonale randen duidelijk. Coiflets zijn een bepaald soort orthonormale scaling functions en wavelets. Voor meer informatie over welke soorten (orthonormale) scaling functions en wavelets er bestaan kan men (Misiti, Misiti, Oppenheim, & Poggi, 2008) of (Daubechies, 1992) raadplegen. Het voorbeeld van Figuur 40 en Figuur 41 is gebaseerd op (Mallat, 1989).



Figuur 38: testfiguur, een standaard beeldverwerkingfoto van een vrouw met een kap.



Figuur 39: 2D niet-standaard wavelet decompositie (2 iteraties) van Figuur 38 met de Box scaling function en Haar wavelet.



Figuur 40: testfiguur, een rechthoek met afgeronde hoeken. Aan de afgeronde hoeken ontstaan een soort van diagonale randen die we kunnen detecteren met een 2D wavelet decompositie.



Figuur 41: 2D niet-standaard decompositie (5 iteraties) van Figuur 40 met de *Coiflet* scaling function en wavelet (type 1).

Voor de volledigheid willen we benadrukken dat het zowel bij de 2D standaard als de 2D niet-standaard decompositie mogelijk is om eender welke orthonormale scaling function en bijhorende wavelet te gebruiken.

7.2. Image Querying

Tegenwoordig bestaat er een grote overvloed aan digitale afbeeldingen, welke kunnen worden opgeslagen in grote databases. Het kan zeer interessant zijn voor gebruikers om te zoeken naar bepaalde afbeeldingen. Deze paragraaf beschouwt net zoals de vorige enkel afbeeldingen die bestaan uit bitmap-data.

De gebruiker kan proberen met keywords te zoeken naar een bepaalde afbeelding in een grote database. Dit vereist echter dat elke afbeelding (handmatig) wordt geannoteerd met tags. Bovendien zijn sommige visuele aspecten van een afbeelding lastig om tekstueel te beschrijven.

In deze paragraaf wordt een Image Querying methode besproken waarbij de gebruiker een lage resolutie afbeelding of tekening als example aan de database kan aanbieden. De taak van het zoekalgoritme bestaat er vervolgens in om alle afbeeldingen in de database terug te vinden die "lijken" op deze example. In database terminologie heet deze example de *query (image)*. We zullen in deze paragraaf ook meer de Engelstalige term *image* in de context van het zoekalgoritme gebruiken dan het Nederlandse *afbeelding*.

Veronderstel bijvoorbeeld dat een bepaalde firma zich specialiseert in het aanbieden van zeer hoge resolutie afbeeldingen van schilderijen. Stel ook dat een kunstliefhebber in een willekeurig tijdschrift een kleine afbeelding van een bepaald schilderij ziet staan en dit schilderij graag digitaal aan zeer hoge resolutie wenst te bekijken. Hij zal willen controleren of de bovengenoemde firma een hoge resolutie versie van dit schilderij in zijn database heeft zitten, en indien dat het geval is het schilderij aan te kopen. De kunstliefhebber kan hiertoe de kleine afbeelding uit het tijdschrift zonder moeite inscannen en vervolgens aanbieden aan de reusachtige database van de gespecialiseerde firma. Met het juiste zoekalgoritme zal de kunstliefhebber snel te weten kunnen komen of zijn schilderij (of een zeer gelijkaardige variant) voorkomt in de database.

De query *image* kan daarnaast ook een tekening zijn waarin de gebruiker met ruwe vormen de kleur en schikking van belangrijke visuele elementen aangeeft. Onafhankelijk van hoe hij tot stand komt zal de query in onderstaande tekst een bitmap-afbeelding zijn die door het zoekalgoritme moet verwerkt worden. In dit hoofdstuk beperken we ons tot bitmap-afbeeldingen als queries, terwijl in hoofdstuk 10 ook een andere manier wordt besproken om visuele queries te stellen aan een database, echter niet in de context van wavelets.

Formeel biedt de gebruiker een query Q aan aan de database en heeft daarbij een bepaalde target *image* T in gedachten die hij wil terugvinden. We zullen een *afstand* moeten definiëren tussen de query en een *image* in de database: hoe groter deze afstand, hoe meer de query en de *image* in de database van elkaar verschillen. Het doel van het zoekalgoritme is om redelijk vlug de *images* in de database op te zoeken waarvan de afstand tot de query het kleinst is.

De query Q kan natuurlijk (lichtjes) verschillen van de target T . De query kan bijvoorbeeld een afbeelding van lage resolutie zijn of bepaalde gekleurde vormen en hun onderlinge posities kunnen lichtelijk verschillen indien de query een tekening is van de gebruiker. Ook op de kleur van de query en de target kan onderlinge variatie zitten. De volgende definitie is gebaseerd op (Chávez, 2001).

Definitie: Metriek

Veronderstel een universum U van objecten. De functie $d : U \times U \rightarrow \mathbb{R}$ wordt een metriek genoemd indien aan de volgende eisen voldaan is:

- 1) $\forall x, y \in U : d(x, y) = 0 \Leftrightarrow x = y$ (positiviteit)
- 2) $\forall x, y \in U : x \neq y \Rightarrow d(x, y) > 0$ (strikte positiviteit)
- 3) $\forall x, y \in U : d(x, y) = d(y, x)$ (symmetrie)
- 4) $\forall x, y, z \in U : d(x, y) \leq d(x, z) + d(z, y)$ (driehoeksongelijkheid)

Het paar (U, d) wordt *metrische ruimte* genoemd. Wanneer in 1) enkel de richting \Leftarrow geldt, spreken we van een *pseudo-metriek*.

Men zou de volgende voorbeeld-metrieken op afbeeldingen kunnen definiëren (per kleurenkanaal):

$$\|Q - T\|_1 = \sum_{l,k} |Q_{(l,k)} - T_{(l,k)}|$$

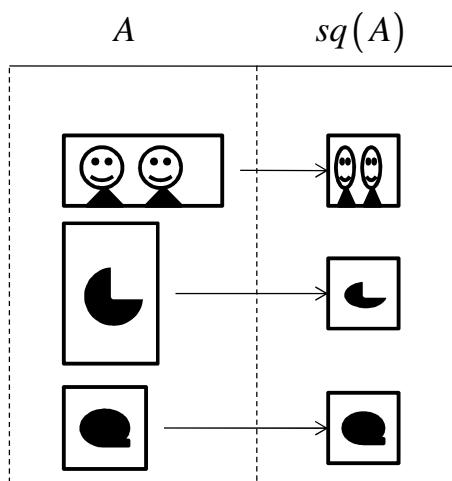
$$\|Q - T\|_2 = \left(\sum_{l,k} (Q_{(l,k)} - T_{(l,k)})^2 \right)^{1/2}$$

Hierbij staat $Q_{(l,k)}$ voor het sample op rij l en kolom k van grid Q (gelijkaardig voor $T_{(l,k)}$). $\|\cdot\|_1$ staat voor de L1-norm en $\|\cdot\|_2$ voor de L2-norm.

Het gebruik van deze voorbeeld-metrieken zou zeer tijdrovend zijn omdat de query image moet vergeleken worden met elke image in de database en bovendien ook sample-per-sample. Door de afwijkingen tussen de query en de target zijn deze metrieken bovendien ook niet accuraat omdat zij vlug een grote afstand berekenen tussen een afbeelding van slechte kwaliteit en zijn tegenpool van goede kwaliteit waarin meer detail aanwezig is.

We zullen vervolgens een image querying metric bespreken die ervoor kan zorgen dat images in de database worden teruggevonden die lichtelijk vervormde versies kunnen zijn van de query.

De query kan gespecificeerd worden op eender welke resolutie, die mogelijk kan verschillen van de images in de database. Omdat de query kan vergeleken worden met de images in de database, kan men alle images, inclusief de query, omzetten naar een bepaalde vaste grootte. Voor de images in de database kan dit gebeuren in een offline preprocessing stap en voor de query kan dit at runtime gedaan worden. In de geraadpleegde literatuur wordt voorgesteld om alle images om te vormen naar een grid-formaat ($N \times N$), dus "vierkante" images. Voor images die een uitgesproken langere zijde hebben zal dit voor een visueel merkbare vervorming zorgen. Deze preprocessing-stap wordt getoond in Figuur 42. We zullen in het vervolg naar deze vierkante images verwijzen als *square-images*. Zij A een oorspronkelijke image, dan noteren we de square-image ervan met $sq(A)$. Hierbij kan men zich $sq(\cdot)$ voorstellen als een operatie die A omzet naar zijn bijhorende square-image. Dit is feitelijk een standaard rescaling van een afbeelding waarbij pixelwaarden kunnen geïnterpoleerd worden via bepaalde technieken. We veronderstellen dat dit mogelijk is op een bepaalde manier waarvan de concrete details niet belangrijk zijn. Merk wel op dat de manier waarom men een afbeelding omzet naar een square-image bepalend kan zijn voor de resultaten die men bekomt met wavelet-analyse, maar daar gaan we niet dieper op in.



Figuur 42: het omzetten van elke afbeelding naar een gemeenschappelijke vierkante grootte.

Vervolgens wordt uit elke square-image een *signature* afgeleid. Dit zijn een aantal getalletjes die redelijk goed de visuele inhoud van een image moeten beschrijven.

De 2D wavelet decompositie is een goede basis voor de signature, want:

- 1) De wavelet decompositie laat toe om met redelijk weinig wavelet coëfficiënten toch een ruwe approximatie te geven van signalen (waaronder images).
- 2) Zoals het voorbeeld in Figuur 39 laat uitschijnen, kunnen we met een wavelet decompositie *rand*-informatie afleiden.
Wanneer de gebruiker met ruwe vormen een tekening maakt en submit als query zal hij vooral de randen in zijn tekening belangrijk vinden omdat die aangeven waar een vorm stopt en een vorm met een andere kleur begint, etc.
- 3) De coëfficiënten van een wavelet decompositie bieden informatie die onafhankelijk is van de resolutie van de originele image. Er zijn natuurlijk wel evenveel coëfficiënten als samples in de input-image, maar men mag niet vergeten dat de coëfficiënten eigenlijk horen bij continue functies in een orthonormale basis. Men maakt als het ware de overstap van "onzuivere" samples in een image-grid naar zuivere continue functies die volledig de image beschrijven.
- 4) Wavelet decompositions zijn efficiënt om te berekenen, zoals aangetoond in hoofdstuk 5.

Er zijn echter nog een aantal andere belangrijke beschouwingen en technieken:

- 1) In welke color space worden de images beschouwd? Afhankelijk van de color space zullen de kanalen andere samples bevatten. In de geraadpleegde literatuur wordt voorgesteld om de color space YIQ te gebruiken. Dit zou experimenteel de beste resultaten geven. Wij zullen geen color space vastleggen om de discussie meer objectief te houden.
- 2) Welke scaling functions en wavelets worden gebruikt? In principe kan men voor de 2D wavelet transformatie eender welk paar samenhangende scaling functions en wavelets gebruiken. De eenvoudigste (en snelste) is de Box-function en bijhorende Haar-wavelet. Bij lossy image compression geeft dit paar wel aanleiding tot visuele artefacten, maar de wavelet decompositie bij image querying wordt enkel gebruikt om een signature te berekenen en wordt nooit getoond aan de gebruiker.
Verder veronderstellen we de niet-standaard 2D wavelet decompositie omdat deze volgens de literatuur het beste werkt om bogen te detecteren van vormen die even hoog zijn als breed. Natuurlijk is dit niet op voorhand geweten, maar de standaard 2D wavelet decompositie zou vooral randen/vormen detecteren die rechthoekig van aard zijn, met ofwel een uitgesproken verticale rand of horizontale rand.
- 3) Hoe groot moeten we de square-images maken? Indien bijvoorbeeld de grootte (128×128) gekozen wordt, dan bevat de wavelet decompositie $128^2 - 1 = 16\,383$ wavelet-coëfficiënten (en 1 approximatie-coëfficiënt), voor één kleurenkanaal. Alle afbeeldingen van de database en de query zelf zullen waarschijnlijk bij praktische toepassingen grotere afmetingen hebben dan (128×128) , dus de bijhorende square-images zijn al een redelijk sterke reductie van de beschikbare informatie in die afbeeldingen. Men moet per database beslissen welke afmetingen voor de square-images nog toelaten om de afbeeldingen voldoende te inhoudelijk (visueel) te onderscheiden.
- 4) In het vorige puntje is vermeld dat zelfs voor kleine afmetingen van de square-images er al veel coëfficiënten kunnen ontstaan. Maar we moeten deze coëfficiënten natuurlijk niet allemaal gebruiken in de signature. Het is mogelijk om een *truncation* uit te voeren: houd enkel de m wavelet-coëfficiënten over met de grootste magnitude. Dit zorgt voor een versnelling bij het vergelijken van signatures omdat signatures dan kleiner worden en

bovendien is er ook minder opslagruimte vereist om de signatures op te slaan. In paragraaf 7.1 hebben we gezien dat na een volledige 2D wavelet-decompositie er slechts één approximatie-coëfficiënt overblijft. De approximatie-coëfficiënt wordt niet weggegooid bij truncation omdat de detail-waarden van de wavelet-coëfficiënten enkel kunnen begrepen worden *bovenop* de benadering die beschreven wordt door de approximatie-coëfficiënt. Zonder de approximatie-coëfficiënt verliezen de geselecteerde wavelet-coëfficiënten een deel van hun betekenis.

Maar hoe houden we concreet deze m wavelet-coëfficiënten en de ene approximatie-coëfficiënt bij? We kunnen bijvoorbeeld voor elke coëfficiënt zijn waarde bijhouden maar ook zijn x - en y -coördinaat in de 2D wavelet-decompositie. We krijgen in dat geval dus niet een lijst van getallen, maar een lijst van drietallen.

In 3.1 hebben we gezien dat geschaleerde scaling functions (en later ook wavelets) worden *genormaliseerd* zodat hun norm gelijk is aan die van de moeder scaling function (en wavelet).

Hoogfrequente componenten in een signaal worden gemodelleerd met dunne scaling functions en wavelets. Omdat deze zo dun zijn zorgt de normalisatie ervoor dat zij ook redelijk hoog zijn. De hoogte van deze dunne functies kan veel hoger zijn dan de samplewaarden in een image, waardoor de wavelet-coëfficiënten die horen bij deze dunne functies eigenlijk zeer laag moeten zijn. Indien men dan de lage wavelet-coëfficiënten vermenigvuldigt met de hoge amplitude van de bijhorende dunne scaling function of wavelet, bekomt men een stukje van een sample uit de image. In de 2D situatie betekent een "dun" dus een kleine oppervlakte-support.

In een 2D wavelet decompositie horen de coëfficiënten van de 2D scaling function in de linkerbovenhoek bij lage resolutie componenten in het signaal. De support van de 2D scaling function bij een volledige decompositie is zo groot als de oorspronkelijke afbeelding, waardoor zijn amplitude wegens de normalisatie erg laag is. Daarom moet de ene 2D approximatie coëfficiënt een grote magnitude hebben om deze lage amplitude te compenseren. De direct omliggende lage resolutie 2D wavelets hebben een even grote support als de 2D scaling function en dus ook een lage amplitude. Om dit te compenseren moeten de wavelet coëfficiënten ook weer een extra grote magnitude hebben.

We voelen duidelijk aan waar de grootste m coëfficiënten zich zullen bevinden in de gedecomponeerde square-images: in de linkerbovenhoek. Er zijn meestal uitgesproken verschillen tussen de magnitudes van coëfficiënten die horen bij zeer lage resolutie componenten in de square-image en de magnitudes van coëfficiënten die horen bij zeer hoge resolutie componenten.

Door de wavelet-coëfficiënten over te houden met de grootste amplitude houden we dus enkel de meest prominente kenmerken van een image over en negeren we de fijnere details. Dit is wenselijk omdat de query image meestal van veel lagere kwaliteit zal zijn en weinig tot geen details zal bevatten. De wavelet-coëfficiënten in de query image zullen zelfs zonder truncation al zeer klein (of zelfs nul) zijn voor de hoge-resolutie componenten in het signaal. Door de truncation uit te voeren op wavelet decomposities van de square-images in de database worden de signatures in de database eigenlijk op "gelijke hoogte" gebracht met de signature van de query.

Voor elke square-image houden we na truncation precies m coëfficiënten over. De precieze dimensies van de square-images is niet heel belangrijk, zolang er maar "genoeg" wavelet-coëfficiënten kunnen berekend worden om prominente kenmerken in een square-image te beschrijven. Vanaf nu veronderstellen we dat de square-images een niet nader bepaalde grootte ($N \times N$) hebben.

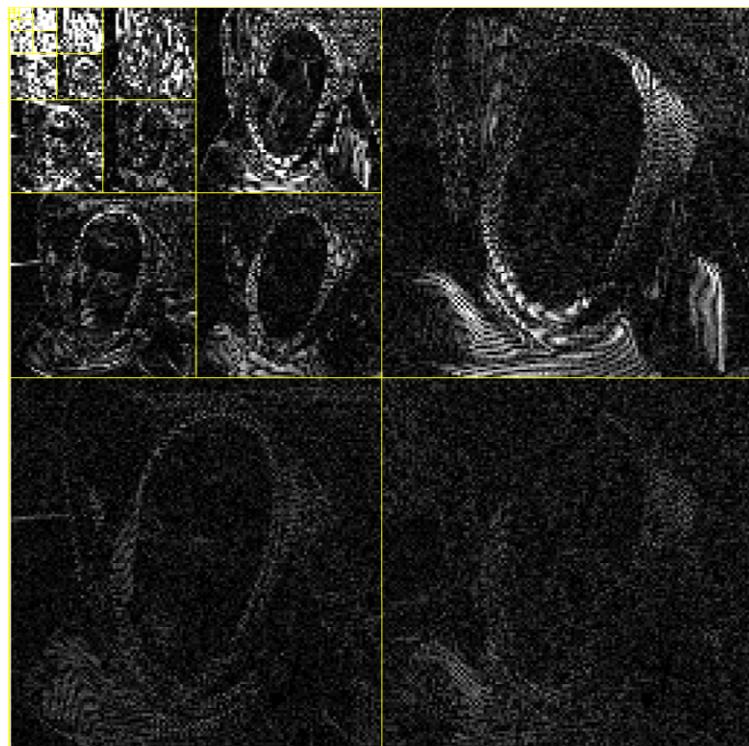
Nogmaals, het is zeer belangrijk om te beseffen dat de approximatie-coëfficiënt nooit door truncation wordt weggegooid in de besproken image querying techniek.

- 5) Na de truncation van de wavelet-coëfficiënten is het ook mogelijk om een *quantization* (quantisering) uit te voeren op de geselecteerde m wavelet-coëfficiënten. De gequantiseerde coëfficiënten houden bijna geen informatie meer bij over de precieze magnitude van de truncated coëfficiënten: enkel het teken wordt overgehouden.
 Bijvoorbeeld: -15 wordt omgezet naar -1 en 330 wordt omgezet naar 1 .
 Door de quantisering wordt nogmaals de tijd om twee signatures te vergelijken versneld en moet nog minder signature-informatie worden opgeslagen.

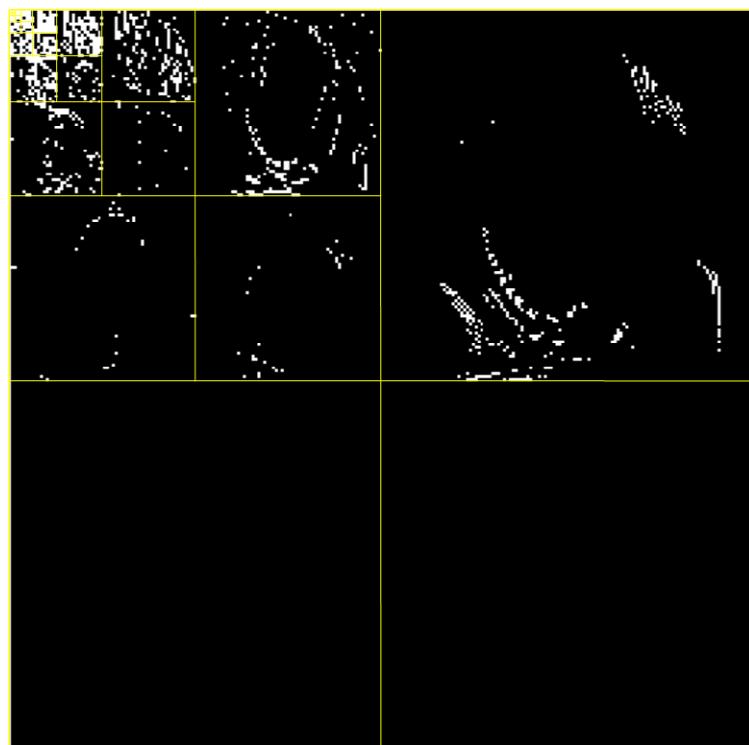
Zij $sq(A)$ de square-image van image A . We zullen nog wat extra notatie invoeren:

- Met $w \circ sq(A)$ duiden we de volledige 2D niet-standaard wavelet decompositie van de samples in $sq(A)$ aan. De wavelet decompositie gebeurt *per kleurenkanaal* afzonderlijk. Omdat we bij de 2D wavelet decompositie verondersteld hebben dat er niet meer coëfficiënten worden gemaakt dan er samples zijn is $w \circ sq(A)$ zelf ook een square-image (met evenveel kleurenkanalen als $sq(A)$).
- Zij t_m de operator die uit $w \circ sq(A)$ de approximatie-coëfficiënt selecteert en daarbij ook de m wavelet-coëfficiënten met de grootste magnitude voegt. Operator t_m voert dus de *truncation* uit. Zij q de quantiserings-operator die alle geselecteerde wavelet-coëfficiënten quantiseert en de approximatie-coëfficiënt ongemoeid laat.
 Met $q \circ t_m \circ w \circ sq(A)$ duiden we de truncated en gequantiseerde versie aan van $w \circ sq(A)$. Ook dit gebeurt per kleurenkanaal afzonderlijk. De ene approximatie-coëfficiënt per kleurenkanaal wordt dus steeds geselecteerd en nooit gequantiseerd.

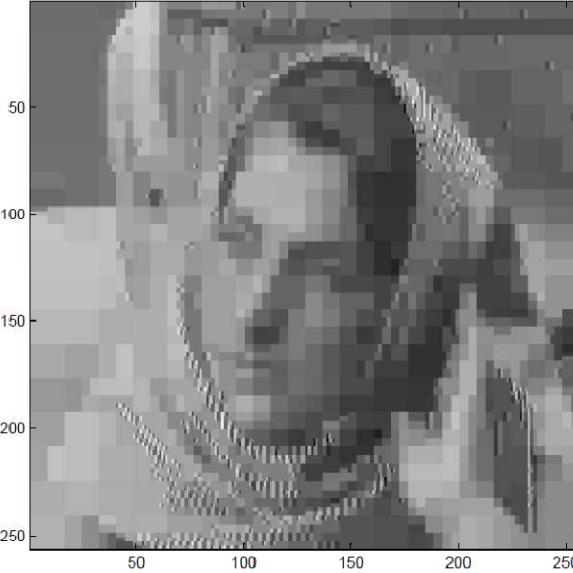
Om een idee te krijgen van wat truncation betekent kan men Figuur 44 bekijken: dit is een truncated versie van de coëfficiënten in Figuur 43. De grootste 623 van de 65536 coëfficiënten werden overgehouden (inclusief de approximatie-coëfficiënt), de rest is zwart gekleurd. Dit is slechts 0.95%. Men kan toch zien dat de belangrijkste vormen van Figuur 38 zijn behouden: de rand van de kap, de omtrek van het gezicht en de schikking van beide vormen in het geheel. Wanneer we de inverse transformatie doen van deze coëfficiënten (2D wavelet-synthese), bekomen we Figuur 45. We kunnen zien dat de basisvormen van de originele afbeelding vrij goed behouden zijn. Voor elke square-image zal men via de signature enkel dit soort visuele basisvormen kunnen beschrijven, maar deze lijken meestal genoeg informatie te bevatten om afbeeldingen visueel te onderscheiden, hoewel dit zal afhangen van de concrete situatie. Op de 2D wavelet-synthese zullen we echter niet dieper ingaan omdat de nadruk in dit hoofdstuk enkel ligt op het analyse-gedeelte. Voor meer informatie over de 2D wavelet-synthese algoritmen voor zowel de standaard als niet-standaard algoritmen met de Box scaling function en Haar wavelet kan men (Stollnitz, Derose, & David, 1996) raadplegen.



Figuur 43: volledige 2D niet-standaard decompositie van Figuur 38 met de Box scaling function en Haar wavelet.



Figuur 44: truncation toegepast op de coëfficiënten in Figuur 43.



Figuur 45: deze afbeelding is het resultaat na het toepassen van een 2D wavelet-synthese op de geselecteerde coëfficiënten in Figuur 44.

We zullen $q \circ t_m \circ w \circ sq(A)$ de *getransformeerde square-image* noemen. Zeer belangrijk: de similarity tussen twee oorspronkelijke images A en B wordt uitgedrukt via een afstandsfunctie tussen de getransformeerde square-images van A en B . Hoe groter de afstand tussen $q \circ t_m \circ w \circ sq(A)$ en $q \circ t_m \circ w \circ sq(B)$, hoe lager de similarity tussen A en B zelf. Deze afstandsfunctie wordt geconstrueerd in de volgende paragraaf.

Belangrijk is dat voor álle square-images *dezelfde* scaling function en bijhorende wavelet gebruikt worden in de volledige 2D niet-standaard wavelet decompositie. De scaling function en wavelet worden op voorhand vastgelegd. Afhankelijk van deze keuze zullen de wavelet-coëfficiënten andere waarden aannemen. Indien specifiek geweten is voor welke soort images de database zal gebruikt worden kan men de scaling function en wavelet zodanig kiezen dat de berekende coëfficiënten zeer goed de kenmerken van de inhoud van dat soort images weergeeft. Daar gaan we niet dieper op in.

7.2.1. Image querying afstandsfunctie

We definiëren in deze paragraaf een afstandsfunctie tussen twee *getransformeerde square-images* Q en T , beiden van grootte $(N \times N)$. Square-image Q noemen we de query en square-image T de target. We veronderstellen voorlopig dat er slechts één kleurenkanaal is zodat Q en T beiden een simpel grid van waarden zijn.

$Q_{(0,0)}$ en $T_{(0,0)}$ duiden de approximatie-coëfficiënten aan van Q respectievelijk T .

Verder duiden $\hat{Q}_{(l,k)}$ en $\hat{T}_{(l,k)}$ de truncated en gequantiseerde wavelet coëfficiënt aan in grid-hokje (l,k) van Q respectievelijk T . Bijgevolg kunnen $\hat{Q}_{(l,k)}$ en $\hat{T}_{(l,k)}$ slechts de waarden -1, 0 en 1

aannemen. Om een gemakkelijke notatie te bekomen in sommaties stellen we $\hat{Q}_{(0,0)}$ en $\hat{T}_{(0,0)}$ gelijk aan nul.

We definiëren een image querying metriek:

$$\|Q - T\| = w_{0,0} |Q_{(0,0)} - T_{(0,0)}| + \sum_{l,k} w_{l,k} (\hat{Q}_{(l,k)} \neq \hat{T}_{(l,k)}) \quad (7.1)$$

Hierbij levert expressie $a \neq b$ de waarde 1 op indien a niet gelijk is aan b en anders levert $a \neq b$ de waarde 0 op.

We zien dat er gewichten $w_{l,k}$ staan in uitdrukking (7.1). Deze kunnen gebruikt worden om bepaalde termen meer invloed te geven in deze metriek. Hiermee kunnen andere afstandsmaten gedefinieerd worden voor specifieke databases of querying-stijlen. Het is een feit dat $\|Q - T\|$ klein is indien square-images Q en T erg similar zijn en dat $\|Q - T\|$ eerder groot is indien dit niet het geval is. In het bijzonder: indien $Q = T$ zal gelden dat $|Q_{(0,0)} - T_{(0,0)}| = 0$ en ook dat $(\hat{Q}_{(l,k)} \neq \hat{T}_{(l,k)}) = 0$ voor alle l en k , waardoor $\|Q - T\| = 0$.

We kunnen ook enkel de termen in de sommatie beschouwen waarvoor de query-coëfficiënten verschillend zijn van nul. Hierdoor is (7.1) sneller te berekenen en bovendien kan hierdoor een query gematcht worden met een target image waar detail-coëfficiënten in aanwezig zijn die niet in de query aanwezig zijn. We brengen geen ongematchte coëfficiënten in rekening die wel in de target aanwezig zijn maar niet in de query. Maar we brengen wel de ongematchte coëfficiënten in rekening van de query die niet aanwezig zijn in de target. Dit is dus een duidelijke asymmetrie, waardoor onze metriek geen metriek meer is volgens de strikte definitie. We zullen daarom nu de term afstandsfunctie gebruiken.

Is deze asymmetrie een slechte zaak? We mogen niet vergeten wat het oorspronkelijke doel was van het image querying algoritme zoals besproken aan het begin van paragraaf 7.2: het doorzoeken van een database van hogere resolutie images die lijken op een lage resolutie query-image. Deze lage resolutie query-image kan een ingescande miniversie zijn van een hogere resolutie image, of een JPEG-image van een webpage waarvan de oorspronkelijke details zijn verdwenen door een hoge compressie-factor, of zelfs een handgetekende afbeelding. De lage resolutie query-image bevat dus feitelijk (over het algemeen) veel minder informatie over details dan de images in de database zelf. De toegevoegde asymmetrie aan onze afstandsfunctie is dus een zeer goede zaak omdat hij ons in staat stelt een afstand te definiëren tussen de query-image en de images in de database waarbij de query-image niet wordt "afgestraft" omdat hij enkel maar informatie bevat over ruwe vormen en kleuren. De asymmetrie komt dus de hele werking van het zoekalgoritme ten goede.

De afstandsfunctie wordt nu:

$$\|Q - T\|_q = w_{0,0} |Q_{(0,0)} - T_{(0,0)}| + \sum_{l,k: \hat{Q}_{(l,k)} \neq 0} w_{l,k} (\hat{Q}_{(l,k)} \neq \hat{T}_{(l,k)}) \quad (7.2)$$

De meeste database-images zullen waarschijnlijk niet goed overeenkomen met de query en daarom zullen veel coëfficiënten van de query niet worden teruggevonden bij de images in de

database. De berekening van de afstandsfunctie kan dus nog sneller gaan wanneer we de coëfficiënten zouden tellen die wél overeenkomen in plaats van de coëfficiënten die niet overeenkomen. Daarom herschrijven we de uitdrukking

$$\sum_{l,k:\hat{Q}_{(l,k)} \neq 0} w_{l,k} (\hat{Q}_{(l,k)} \neq \hat{T}_{(l,k)})$$

Stel dat de uitdrukking $a = b$ de waarde 1 oplevert indien a gelijk is aan b en anders 0. Er geldt dat de uitdrukking $a \neq b$ precies dezelfde waarden oplevert als $1 - (a = b)$. Daarom geldt:

$$\sum_{l,k:\hat{Q}_{(l,k)} \neq 0} w_{l,k} (\hat{Q}_{(l,k)} \neq \hat{T}_{(l,k)}) = \sum_{l,k:\hat{Q}_{(l,k)} \neq 0} w_{l,k} (1 - (\hat{Q}_{(l,k)} = \hat{T}_{(l,k)}))$$

Dit kan verder uitgewerkt worden naar:

$$\sum_{l,k:\hat{Q}_{(l,k)} \neq 0} w_{l,k} - \sum_{l,k:\hat{Q}_{(l,k)} \neq 0} w_{l,k} (\hat{Q}_{(l,k)} = \hat{T}_{(l,k)}) \quad (7.3)$$

We zullen nu de intuïtie geven achter de herschreven vorm in (7.3). In beide sommaties overlopen we nog steeds enkel de coëfficiënten waarvoor de query een niet-nul waarde heeft. Dit stemt bijgevolg nog steeds overeen met de asymmetrie-constructie die we tevoren hadden geïntroduceerd. Wat wel nieuw is het volgende: de eerste sommatie telt zomaar alle gewichten $w_{l,k}$ op, zonder te kijken of de query en de target overeenkomende coëfficiënten hebben of niet. Indien we enkel deze eerste sommatie zouden beschouwen wordt uiteraard de afstand tussen de query en de target groter dan nodig. Daarom worden in de tweede sommatie alle gewichten $w_{l,k}$ opgeteld die horen bij coëfficiënten die overeenkomen tussen de query en de target. Deze tweede sommatie wordt van de eerste afgetrokken zodat globaal gezien de hele expressie (7.3) enkel de gewichten optelt van coëfficiënten die aanwezig zijn in de query maar niet in de target. De hele herschreven afstandsfunctie is nu:

$$\|Q - T\|_q = w_{0,0} |Q_{(0,0)} - T_{(0,0)}| + \sum_{l,k:\hat{Q}_{(l,k)} \neq 0} w_{l,k} - \sum_{l,k:\hat{Q}_{(l,k)} \neq 0} w_{l,k} (\hat{Q}_{(l,k)} = \hat{T}_{(l,k)}) \quad (7.4)$$

Maar aangezien de eerste sommatie enkel afhankelijk is van de query square-image en niet van de target, mogen we deze sommatie in principe weglaten uit de berekening van de afstandsfunctie. We kunnen immers door het uitrekenen van de tweede sommatie toch de images in de database rangschikken op similarity met de query. Maar zonder de eerste sommatie kan de afstand ook negatief worden natuurlijk. In de pseudo-code van het zoekalgoritme dat we hierna zullen zien werd ervoor gekozen om toch de eerste sommatie te behouden in de berekening.

De afstandsfunctie tussen twee getransformeerde square-images Q en T is dus een gewogen som van het verschil in gemiddelde kleur (approximatie-coëfficiënten) en het aantal truncated en gequantiseerde wavelet coëfficiënten uit T waarvan het teken en de indices overeenkomen met die van Q .

7.2.2. Image querying algoritme

We zullen nu het zoekalgoritme bespreken om gegeven een query-image alle images in de database terug te vinden die het meest gelijken op deze query. Merk op hoe het probleem geformuleerd wordt: we willen “de meest gelijkende” images. Indien er in de database enkel images zijn waarvan wij als mensen visueel beoordelen dat zij helemaal niet op de query lijken, moet er toch een ordening van deze database-images gemaakt worden met de meest gelijkende images vanvoren. De notie “meest gelijkend” werd objectief vastgelegd in de afstandsfunctie van de vorige paragraaf. We hebben het probleem dus niet geformuleerd als range-searching waarbij enkel de images in de database worden teruggegeven waarvan de afstand tot de query in een bepaalde error-marge ligt.

De tijdscomplexiteit van het algoritme dat hieronder wordt besproken is lineair in functie van het aantal images in de database. Maar we zullen zien dat het geen sequential search is. Er is in feite helemaal geen sprake van searching omdat voor alle afbeeldingen in de database een afstand berekend wordt tot de query-image (gebaseerd op de signatures van de square-images). Indexatie-structuren worden in dit geval niet gebruikt om de zoekruimte te verkleinen maar om efficiënt te weten welke afbeeldingen in de database bepaalde wavelet-coëfficiënten hebben.

Herinner u dat de square-images grootte $(N \times N)$ hebben. Vanaf nu veronderstellen we dat elke image I (inclusief de query) C kleurenkanalen heeft. Meestal zijn dit er drie. Het c^{de} kleurenkanaal van image I noteren we met I^c .

In onderstaande tekst duiden we met A de niet-getransformeerde (oorspronkelijke) query-image aan die de gebruiker aanbiedt aan de database. Stel $Q^c = q \circ t_m \circ w \circ sq(A^c)$, dan noteren we met $Q_{(0,0)}^c$ de approximatie-coëfficiënt in Q^c en met $\hat{Q}_{(l,k)}^c$ de wavelet-coëfficiënten in Q^c .

We zullen nu beschrijven hoe er een indexatie-structuur kan gebouwd worden over alle images van de database. We noemen θ^{c+} de *positieve search grid* voor kleurenkanaal c en θ^{c-} de *negatieve search grid* voor kleurenkanaal c . Indien er drie kleurenkanalen zijn hebben we zes verschillende search grids.

Elk hokje (l,k) in θ^{c+} bevat een lijst van alle (identifiers van) images T in de database die een waarde 1 hebben in $\hat{T}_{(l,k)}^c$. Elk hokje (l,k) in θ^{c-} bevat een lijst van alle (identifiers van) images T in de database die een waarde -1 hebben in $\hat{T}_{(l,k)}^c$. Het berekenen van alle square-images, wavelet decomposities, truncations, quantiseringen en tenslotte het opbouwen van de genoemde search grids kan allemaal offline gebeuren.

Nu kunnen we beschrijven hoe de querying gebeurt at runtime. De gebruiker biedt een image A aan het zoekalgoritme. De pseudo-code van het algoritme is weergegeven in Listing 1. Hierbij hebben we een aantal nieuwe symbolen ingevoerd:

- D : het aantal images in de database
- $w_{l,k}^c$: gewicht $w_{l,k}$ uit vergelijking (7.4), maar dan voor kleurenkanaal c

- $dist$: een array of tabel waar voor elke identifier van een image in de database wordt bijgehouden wat zijn afstand is tot de query image

Listing 1: Image Querying zoekalgoritme

Constants:

- C : number of color channels
- size $(N \times N)$ of each square-image
- $m \in \mathbb{N}$: number of wavelet coefficients kept at the truncation-step

Input: query-image A

$$\forall i = 1 \dots D : dist[i] \leftarrow 0$$

$$\forall c = 1 \dots C : Q^c = q \circ t_m \circ w \circ sq(A^c)$$

for $c = 1 \dots C$ **do**

for each database image T **do**

$$dist[id(T)] \leftarrow dist[id(T)] + w_{0,0}^c |Q^c_{(0,0)} - T^c_{(0,0)}| + \sum_{l,k: \hat{Q}_{(l,k)}^c \neq 0} w_{l,k}^c$$

end for

[*] **for** each nonzero, truncated, quantised wavelet coefficient $\hat{Q}_{(l,k)}^c$ **do**

if $\hat{Q}_{(l,k)}^c > 0$ **then**

$list \leftarrow \theta_{(l,k)}^{c+}$ // positive search grid of color c

else

$list \leftarrow \theta_{(l,k)}^{c-}$ // negative search grid of color c

end if

for each image T in $list$ **do**

$dist[id(T)] \leftarrow dist[id(T)] - w_{l,k}^c$

end for

end for

end for

Het algoritme berekent voor elke image T de afstand tussen de getransformeerde square-image van T en de square-image van A . Alle berekende afstanden worden onthouden in de array $dist$. Het is de bedoeling dat deze later door het database-programma wordt *gesorteerd* met de kleinste afstandswaarde bovenaan. Aan de gebruiker zullen images getoond worden in deze volgorde. Het beschouwen van meerdere kleurenkanalen is een eenvoudige uitbreiding van (7.4). Voor elk kleurenkanaal c en elke image T voeren we uit:

$$dist[id(T)] \leftarrow dist[id(T)] + w_{0,0}^c |Q_{(0,0)}^c - T_{(0,0)}^c| + \sum_{l,k: \hat{Q}_{(l,k)}^c \neq 0} w_{l,k}^c$$

De sommatie die achteraan wordt bijgeteld is afkomstig van de herschrijving in (7.3).

De tijdscomplexiteit van het zoekgoritme is lineair in functie van D omdat C , N en m constant zijn per database-instantie. Afhankelijk van de query-image zal for-loop [*] veel of weinig iteraties uitvoeren per kleurenkanaal. Aan C is meestal niet veel te veranderen, omdat dit meestal drie of vier is. Maar indien N en m eerder klein zijn zal for-loop [*] niet veel werk hebben. Het is mogelijk dat alle *list*-variabelen een lege lijst bevatten indien de query zeer slecht lijkt op de images in de database! In dat geval bevatten alle entries in *dist* de maximale afstand tot de query.

Tenslotte merken we nog op dat de auteurs in (Stollnitz, Derose, & David, 1996) voorstellen om de gewichten $w_{l,k}^c$ onder te verdelen in een beperkt aantal "bins" waardoor minder werk vereist is om de afstandsfunctie te *tunen*.

Alle gewichten in de gebruikte formules zijn parameters die ingesteld moeten worden door een gebruiker. Maar het is natuurlijk niet eenvoudig om als mens te beslissen over de precieze waarden. Daarvoor kan eventueel een machine learning algoritme gebruikt worden waarbij we als input kunnen specificeren welke afbeeldingen gelijkaardig zijn en waaruit automatisch de waarden voor de gewichten kunnen bepaald worden. Elk paar gelijkaardige afbeeldingen is dan een constraint op de gewichten. Daar gaan we momenteel niet dieper op in.

8. Korte inleiding tot similarity technieken

8.1. Onderscheid kwantitatief-kwalitatief maatstaven

Er is een belangrijk onderscheid tussen kwantitatieve en kwalitatieve maatstaven die gebruikt kunnen worden om similarity uit te drukken tussen data-objecten. Dit onderscheid is eigenlijk niet ingewikkeld, maar toch nuttig om kort te bespreken.

Bij kwantitatieve methoden worden wiskundige formules gebruikt om getallen uit te rekenen die uitdrukken hoe goed twee objecten op elkaar lijken. Dit is meestal gebaseerd op domein-specifieke regels, zoals bijvoorbeeld het verschil tussen de samples van een dasequentie (Kellens, 2006), (Keogh, Wei, Xi, Lee, & Vlachos, 2006). Meestal volgt men het principe: hoe kleiner het berekende getal, hoe meer de twee objecten op elkaar lijken. De formules rekenen dus een soort van afstand uit, die vaak niet voldoet aan de driehoeksongelijkheid en daardoor zeker geen metriek is. Het voordeel van kwantitatieve methoden is de precisie want er kan eventueel zeer accuraat een verschil berekend worden tussen objecten wanneer zij reële getallen als attributen hebben (Gottfried, 2008).

Bij kwalitatieve technieken worden kenmerken van objecten of hun onderlinge relaties in ruwe categorieën geplaatst. Een voordeel van het gebruik van ruwe categorieën over de kwantitatieve methoden is dat zij veel compactere beschrijvingen toelaten en vaak ook aanleiding geven tot relatief snelle implementaties (Egenhofer, 1997). Bovendien zijn ze minder gevoelig voor noise op de data omdat noise slechts op relatief kleine schaal vervormingen aanbrengt waardoor de ruwe opdeling in categorieën niet drastisch verandert. Een ander voordeel van kwalitatieve maatstaven is dat zij soms eenvoudiger door mensen kunnen geïnterpreteerd worden op een conceptuele manier (Gottfried, 2008), in een user interface bijvoorbeeld. We kunnen als mens eenvoudig oordelen dat twee punten in het vlak allebei aan dezelfde kant liggen van een lijn, maar niet een precies getal plakken op de kortste afstand van de punten tot die lijn of tot elkaar.

Bij kwantitatieve methoden is een externe schaal nodig, zoals Euclidische afstand in het vlak of het meten van een hoek. Kwalitatieve methoden daarentegen beschouwen enkel eigenschappen door het vergelijken van de categorieën van de objecten en hun relaties. In (Gottfried, 2008) wordt over kwalitatieve methoden gezegd: "*what distinguishes them is that they rely on differences in kind, rather than of measurement*".

Een typische eigenschap van kwalitatieve methoden is echter dat zij niet eenvoudig toelaten om de oorspronkelijke objecten terug te reconstrueren aan de hand van hun kwalitatieve beschrijving. De precisie van deze methoden is bijgevolg beperkt.

Zie (Van Hoof, 2007) voor een bespreking over het gebruik van kwantitatieve en kwalitatieve technieken in de context van trajectories.

Tenslotte nog volgende opmerking. We kunnen van een kwantitatieve maat steeds een kwalitatieve maat maken door de range van waarden van de kwantitatieve maat op te splitsen. Dit kan bijvoorbeeld gebeuren door te testen of een waarde tussen bepaalde grenswaarden ligt, waardoor de oorspronkelijke range wordt opgedeeld in bins. Elke bin is dan een aparte categorie.

8.1.1. Voorbeeld kwalitatieve maat

We bespreken in deze paragraaf een voorbeeld van een kwalitatieve maatstaf in de context van GIS.

We gebruiken predicaten P om de binaire relatie tussen twee deelverzamelingen A en B van \mathbb{R}^2 te benoemen. Een predicaat $P(A, B)$ is *topologisch* wanneer het invariant is onder een

topologische transformatie (homeomorfisme). Een topologische transformatie is een bijectie tussen twee deelverzamelingen van het vlak die continu is en waarvan de inverse ook continu is. Continuïteit betekent dat wanneer twee punten in het domein oneindig dicht bij elkaar liggen, hun beelden ook oneindig dicht bij elkaar zullen liggen.

We beperken ons hieronder tot *eenvoudige (simpiele) veelhoeken* als deelverzamelingen van \mathbb{R}^2 . Een veelhoek is eenvoudig wanneer zijn omtrek zichzelf niet snijdt. Wanneer we over veelhoeken spreken, bedoelen we steeds eenvoudige veelhoeken.

Voorbeelden van topologische transformaties op veelhoeken zijn schaleringen, rotaties en translaties. Maar het is ook mogelijk dat de transformatie bepaalde punten van de veelhoek op een arbitraire manier verschuift, zolang het resultaat maar een eenvoudige veelhoek is.

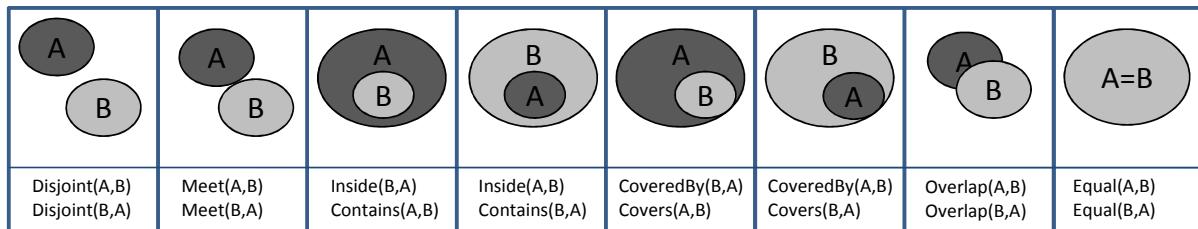
In (Egenhofer, 1997) worden acht high-level topologische relaties tussen twee veelhoeken A en B voorgesteld, genoteerd met volgende binaire predicaten:

- Disjoint
- Meet
- Equal
- Overlap
- Inside/Contains
- Covers/CoveredBy

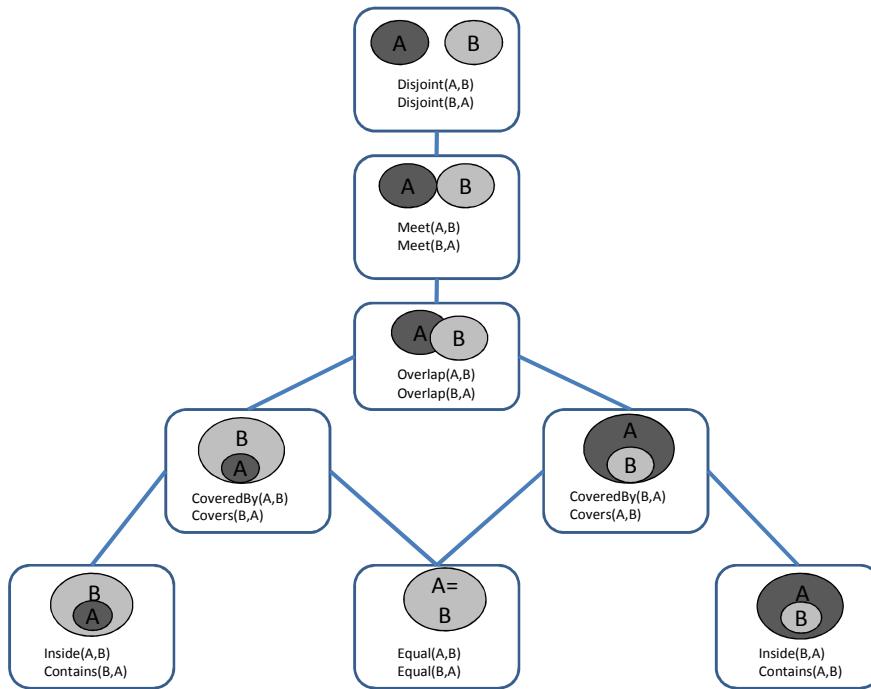
Het predicaat dat achter de "/" staat is de inverse van het predicaat dat vóór de "/" staat. Er zijn natuurlijk ook andere predicaten mogelijk om de relatie tussen twee veelhoeken te benoemen.

Stel bijvoorbeeld dat predicaat $P_k(A, B)$ het volgende betekent: de doorsnede van veelhoek A en B bestaat uit k eenvoudige veelhoeken. Voor elke $k \in \mathbb{N}$ hebben we een ander predicaat, wat aanleiding geeft tot oneindig veel predicaten. Maar we beperken ons in deze tekst voorlopig tot de acht bovennoemde predicaten.

Als we willen uitdrukken dat A raakt aan B schrijven we: $\text{Meet}(A, B)$. Alle high-level relaties en predicaten worden geïllustreerd in Figuur 46.



Figuur 46: ruwe topologische relaties tussen twee veelhoeken zonder gaten.



Figuur 47: conceptual neighborhood graph van high-level topologische relaties

Stel dat we in een query aan een GIS-database de onderlinge relatie van twee veelhoeken A en B zouden beschrijven met één van bovenstaande predicaten. De veelhoeken zelf zijn even niet belangrijk en we concentreren ons op het predicaat van hun binaire relatie.

Stel dat een database een aantal veelhoeken bevat en dat hun binaire relaties benoemd zijn met één van de acht predicaten. We zouden in de database kunnen zoeken naar twee veelhoeken die precies dezelfde topologische relatie hebben als de veelhoeken in de query. We zoeken dan naar een *exacte* overeenkomst tussen het predicaat in de query en een predicaat in de database. Maar natuurlijk kunnen we ook geïnteresseerd zijn in onderlinge relaties van veelhoeken die “lijken” op die van de query. Dit houdt in dat we de query *versoepelen*. Hiertoe beschouwen we de *conceptual neighborhood graph*. De knopen in deze graaf zijn de bovengenoemde high-level topologische relaties en de bogen geven aan welke relaties aan elkaar verwant zijn, zie Figuur 47. Er wordt een boog getrokken vanuit een relatie R naar de relaties die het minst aantal verschillen hebben ten opzicht van de *9-intersection matrix* van R . Deze matrix biedt de mogelijkheid om op een theoretisch onderbouwde manier via het *uitwendige*, de *rand* en het *inwendige* van een ruimtelijk object te redeneren over onderlinge verbanden tussen twee ruimtelijke objecten. We zullen niet dieper ingaan op deze 9-intersection matrix omdat dit momenteel niet nuttig is voor ons. Zie (Egenhofer & Herring, 1991) voor meer informatie. Het belangrijkste is dat men de conceptual neighborhood graph op de een of andere manier kan definiëren.

Het versoepelen van de topologische relatie houdt in dat we tijdens het zoeken in de database ook de conceptuele buren van het predicaat in de query toelaten tussen twee gevonden database-veelhoeken. Dit is slechts één graad van versoepeling. Bijvoorbeeld, indien de relatie tussen beide veelhoeken in de query gelijk is aan $\text{Meet}(A, B)$, mag de relatie tussen gevonden veelhoeken C en D in de database gelijk zijn aan $\text{Disjoint}(C, D)$ of $\text{Overlap}(C, D)$.

Bij sterkere versoepelingen van een relatie R beschouwt men dan de conceptuele buren van de conceptuele buren van een relatie R , enz.

Tenslotte, in (Egenhofer, 1997) wordt nog aangegeven hoe bovenstaande high-level topologische relaties kunnen uitgebreid worden met gedetailleerde informatie die zelfs kwantitatief kan zijn:

- Hoeveel keren snijdt veelhoek A veelhoek B ?

- Hoe lang zijn de gedeelde randen?
- Wat is de verhouding van de oppervlakten van beide veelhoeken?
- ...

Dit geeft aan dat het zeker niet ondenkbaar is dat kwalitatieve en kwantitatieve maatstaven gecombineerd worden in eenzelfde programma. Men kan op die manier misschien zelfs een krachtiger uitdrukkingsmechanisme bekomen. Voor een binaire relatie tussen twee veelhoeken kunnen er bijvoorbeeld meerdere attributen worden bijgehouden, van ruwe high-level topologie tot reële getallen die extra details kunnen verschaffen. Dit impliceert een sterk *modulair* karakter waarbij elk attribuut bijna volledig onafhankelijke of lichtelijk redundante informatie verstrekkt. De gebruiker of het programma kan dan bepalen welke informatie of combinaties het beste bruikbaar zijn in een bepaalde context. Bijvoorbeeld, in (Egenhofer, 1997) wordt benadrukt dat bij het beantwoorden van een veelhoek-query zoals hierboven eerst alle oplossingen kunnen weggesnoeid worden die sterk verschillende high-level topologische relaties hebben, die dus niet dicht bij elkaar liggen in de conceptual neighborhood graph. Vervolgens kan in de overblijvende oplossingen verder gesnoeid worden op basis van meer gedetailleerde relatie-attributen.

9. A complex network-based approach for boundary shape analysis

De omtrek van een object speelt een zeer belangrijke rol in het herkennen ervan. In de literatuur wordt dit de *shape* van een object genoemd. Het is daarom interessant om objecten te herkennen of te onderscheiden op basis van hun shape.

Dit hoofdstuk is gebaseerd op (Ricardo Backes, 2009) en op (Costa, Rodrigues, Travieso, & Villas Boas, 2007). In dit hoofdstuk bespreken we een zeer recente techniek om shapes te beschrijven met een *descriptor* waarmee classificatie of similarity searching in databases mogelijk is. Concreet is een descriptor in dit hoofdstuk een vector van kenmerken (*feature vector*). Een shape in dit hoofdstuk is eenvoudigweg een verzameling punten in het vlak. We zullen hiernaar ook met de term *patroon* verwijzen.

De paper (Ricardo Backes, 2009) werd uitgekozen om te testen hoe het gebruik van descriptors werkt. We zijn ook in de gelegenheid geweest een simpele implementatie van deze techniek te maken. In paragraaf 9.3 formuleren we ons besluit.

Daarnaast is de uitgekozen paper op het moment van schrijven zeer recent waardoor de auteurs ervan de mogelijkheid hadden om terug te blikken op vele reeds bekende (krachtige) descriptor-technieken en deze te vergelijken met hun nieuwe vondst.

Het construeren van de descriptors in dit hoofdstuk is puur een *kwantitatieve* aangelegenheid.

Het voordeel van descriptors in het algemeen is dat zij offline kunnen berekend worden want tijdens het queryen van een database worden enkel nog maar descriptors gebruikt. Bijgevolg mag het berekenen van een descriptor eventueel gebeuren met een iets rekenintensiever algoritme.

9.1. Inleiding tot Complexe netwerken

Deze paragraaf is gebaseerd op (Costa, Rodrigues, Travieso, & Villas Boas, 2007).

Een *complex netwerk* is een andere naam voor een graaf. De naam "graaf" wordt meer gebruikt in de wiskunde en theoretische informatica, terwijl de term "complex netwerk" ook in andere wetenschappen gebruikt wordt, waaronder sociologie, biologie en fysica. In dit hoofdstuk wordt ook de naam "complex netwerk" gebruikt omdat deze veel voorkomt in de literatuur. Een complex netwerk zullen we noteren met G .

Complexe netwerken die gemaakt zijn naar analogie met echte fenomenen hebben eigenschappen die niet kunnen uitgelegd worden met een random connectiviteit tussen de knopen. Bijvoorbeeld, netwerken die bijvoorbeeld Internet verbindingen tussen routers of hyperlinks voorstellen tussen websites bevatten een soort van community structuur met bepaalde centrale knopen die informatie verdelen of opslaan. Elk complex network heeft eigen topologische kenmerken. De analyse van complexe netwerken hangt daarom af van zogenaamde *metingen* die de meest relevante topologische eigenschappen kunnen uitdrukken. Deze metingen hangen af van de toepassing, dus welke netwerken men beschouwd. Een aantal hiervan worden besproken in paragraaf 9.1.2.

Een reden voor het succes van complexe netwerken is hun flexibiliteit en algemeenheid. Dit komt natuurlijk omdat eerder welke natuurlijke structuur kan weergegeven worden met grafen, onder andere ook structuren die evolueren. Bijvoorbeeld, elke discrete structuur zoals lijsten, bomen, roosters,... kunnen allemaal beschouwd worden als speciale gevallen van grafen.

9.1.1. Graaf-terminologie

We zullen nu kort de gebruikte graaf-terminologie introduceren. Het meeste hiervan zal normaal gezien bekend zijn bij de lezer.

Een *gewogen gerichte graaf* G is een tuple (V, E, ω) . Hierbij is V de verzameling van N knopen en E de verzameling van gerichte bogen. Als laatste is er de mapping $\omega: E \rightarrow \mathbb{R}$ die aan elke gerichte boog een gewicht toekent (label). Elke knoop kan aangeduid worden met zijn identifier $i = 1, 2, \dots, N$. Elke boog is een paar van knopen (i, j) en stelt een verbinding voor van knoop i naar knoop j , en het gewicht van deze verbinding is $\omega(i, j)$. We zullen aannemen dat bogen van een knoop naar zichzelf niet voorkomen, dus $i \neq j$. Volgens deze definitie mogen er maximaal twee bogen zijn tussen elk paar knopen, één voor elke richting. Men kan een gerichte graaf gebruiken om bepaalde verbanden voor te stellen tussen de knopen die slechts in één richting gelden, of waarbij het gewicht afhangt van de richting.

Een gewogen gerichte graaf kan voorgesteld worden d.m.v. een $N \times N$ weight matrix W . Elk element $w_{i,j} = \omega(i, j)$ drukt het gewicht uit van de boog (i, j) . Men kan $w_{i,j} = 0$ interpreteren als het ontbreken van boog (i, j) .

Men kan een *thresholding* operatie uitvoeren om een gewogen gerichte graaf om te zetten naar een ongewogen gerichte graaf. Deze operatie, genoteerd als $\delta_T(W)$, wordt toegepast op elk element van de weight matrix W . Hierbij staat T voor de waarde van de threshold. Opgelet: T is een *bovengrens*. Het resultaat van thresholding is $N \times N$ matrix $A = \delta_T(W)$. De elementen van matrix A worden berekend door de overeenkomende elementen in matrix W te vergelijken met de threshold T :

$$\forall w_{i,j} \in W : \begin{cases} a_{i,j} = 0 & \text{als } |w_{i,j}| > T \\ a_{i,j} = 1 & \text{als } |w_{i,j}| \leq T \end{cases}$$

We kunnen in bovenstaande uitdrukking zien dat de absolute waarde van de gewichten gebruikt wordt bij het vergelijken met de threshold. De resulterende matrix A kan geïnterpreteerd worden als de adjacency matrix van de *ongewogen* gerichte graaf. Wanneer $a_{i,j} = 1$ is boog (i, j) aanwezig en anders niet.

Een gewogen *ongerichte* graaf kan gedefinieerd worden aan de hand van een gewogen *gerichte* graaf. Bij een ongerichte graaf zijn de bogen symmetrisch, dus wanneer een boog (i, j) bestaat, moet boog (j, i) ook bestaan. Bovendien geldt $\omega(i, j) = \omega(j, i)$.

Tenslotte, een gewogen gerichte graaf kan omgezet worden naar een gewogen *ongerichte* graaf door een nieuwe weight matrix W' te definiëren:

$$W' = W + W^T$$

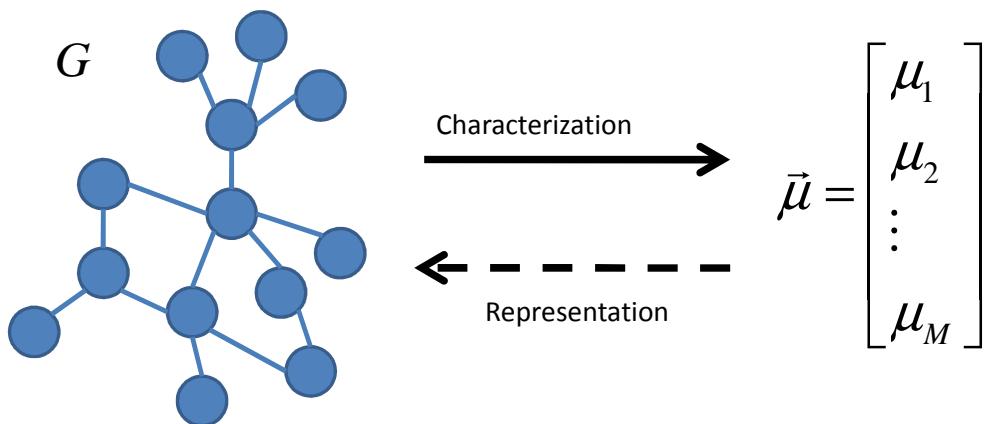
Hierbij staat W^T voor de getransponeerde van matrix W .

9.1.2. Metingen en eigenschappen van complexe netwerken

Het gebruik van complexe netwerken bestaat in het algemeen uit twee stappen:

- 1) Stel de structuur die men wil onderzoeken voor als een complex netwerk
- 2) Analyseer de topologische eigenschappen van het complexe netwerk door “metingen” uit te voeren.

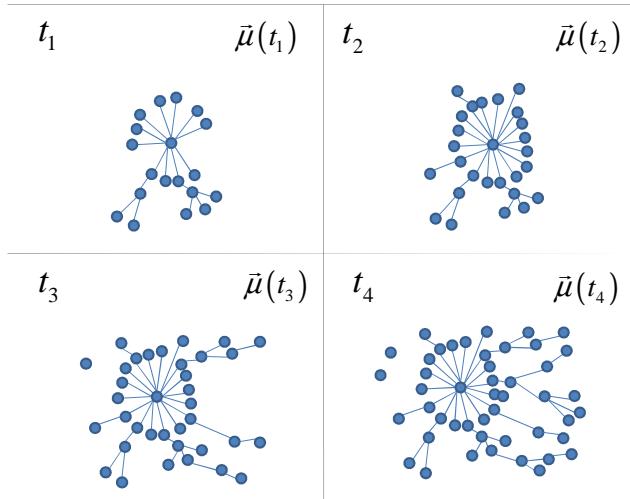
In Figuur 48 kan men schematisch zien hoe een algemeen complex netwerk wordt geprojecteerd naar een feature vector $\vec{\mu}$. Deze feature vector bevat een aantal metingen μ_i ($1 \leq i \leq M$) van de netwerk-topologie die het netwerk *karakteriseren*. Indien de projectie ook een inverse heeft, kan het netwerk terug afgeleid worden uit de feature vector. In dit laatste geval wordt de feature vector ook wel een *representation* genoemd van het netwerk. Een triviaal voorbeeld van een representation is de adjacency matrix. De projectie naar de feature vector die we in dit hoofdstuk bespreken is niet invertibel.



Figuur 48: complex netwerk naar feature vector.

Men kan de feature vectors $\vec{\mu}_a$ en $\vec{\mu}_b$ van twee verschillende complexe netwerken G_a en G_b met elkaar vergelijken. Men kan een verschil-vector $\overrightarrow{\Delta\mu}$ definiëren tussen beide feature vectors (die bijvoorbeeld het components-gewijze verschil $\vec{\mu}_b - \vec{\mu}_a$ voorstelt). Het is wel belangrijk natuurlijk dat beide feature vectors dezelfde metingen in dezelfde volgorde bevatten. Een complex netwerk kan gemanipuleerd worden: voeg bogen toe of verwijder er, voeg knopen toe of verwijder er. Het zou bijvoorbeeld kunnen zijn dat G_b het resultaat is van het manipuleren van G_a .

Men kan de *gevoeligheid* van metingen μ_i beschouwen. Indien kleine veranderingen in de netwerk-topologie grote veranderingen in de metingen veroorzaken (grote waarden van $\|\overrightarrow{\Delta\mu}\|$), dan worden die metingen als erg gevoelig of onstabiel bestempeld.



Figuur 49: evolutie van een complex netwerk. Merk op dat er ook geïsoleerde knopen mogen bijkomen (en verdwijnen).

Het is niet alleen het statische karakter van de knopen en bogen dat interessant is, maar ook de evolutie van de netwerk topologie! Er kunnen bogen en knopen bijkomen en terug verdwijnen doorheen de tijd. Indien het dynamische complexe netwerk natuurgetrouw wordt gesimuleerd, zullen de metingen ervan ook realistische uitspraken mogelijk kunnen maken omtrent het werkelijke fenomeen dat men wenst te bestuderen. Men bekomt een rijkere set van metingen door meerdere netwerk-instanties te beschouwen die ontstaan door een evolutieproces van een startnetwerk. Met $\vec{\mu}(t)$ noteren we de feature vector die wordt verkregen op tijdstip t van het evolutieproces. Afhankelijk van de applicatie kan men feature vectoren maken voor een verschillend aantal tijdstippen. In Figuur 49 wordt een voorbeeldevolutie weergegeven voor tijdstippen $t_1 < t_2 < t_3 < t_4$.

Er is geen eenduidige manier waarop men kan beslissen welke metingen men moet uitvoeren om karakteriserende feature vectoren te bekomen. Verschillende metingen kunnen ongeveer dezelfde topologische eigenschappen beschrijven, maar toch op een lichtelijk andere manier. Uiteindelijk is men aangewezen op eigen kennis en inzicht van de toepassing bij het selecteren van de metingen. In paragraaf 9.2 zijn enkel ongerichte grafen belangrijk als complexe netwerken en daarom zullen we enkel een aantal metingen bespreken voor deze soort.

Beschouw een ongerichte graaf G met N knopen. Zij A de $N \times N$ adjacency matrix van G . Voor ongerichte grafen zijn twee knopen i en j adjacent (naburig) indien $a_{i,j} = 1$. De degree (graad) van een knoop i , genoteerd k_i , wordt gedefinieerd als het aantal knopen dat adjacent is aan i :

$$k_i = \sum_{j=1}^N a_{i,j}$$

Het is nu mogelijk om een aantal metingen te definiëren die gebaseerd zijn op de degrees van de knopen.

De average degree k_μ van een netwerk is het gemiddelde van deze k_i , over alle knopen in het netwerk:

$$k_\mu = \frac{1}{N} \sum_{i=1}^N k_i$$

Een simpele meting is ook de *maximum degree*:

$$k_{\max} = \max \{k_i \mid 1 \leq i \leq N\}$$

In het algemeen is het mogelijk dat twee knopen in een complex netwerk niet adjacent zijn. Veel complexe netwerken die interessant zijn, zijn feitelijk *sparse*: er is slechts een fractie van alle mogelijke bogen aanwezig. Wanneer een netwerk zeer weinig bogen bevat is het mogelijk dat er veel geïsoleerde knopen zijn. Indien we meer bogen toevoegen, zullen er kleine geïsoleerde clusters ontstaan. Bij alsmaar toenemende hoeveelheid bogen zullen de clusters steeds groter worden en zullen clusters ook samensmelten. Uiteindelijk, wanneer alle mogelijke bogen zijn toegevoegd, zullen alle knopen bevatten in één supercluster. Het toevoegen van bogen is slechts een voorbeeld van een evolutieproces dat een complex netwerk kan meemaken. Er kunnen ook bogen verdwijnen of er kunnen zelfs fluctuaties zijn van de hoeveelheden bogen en knopen. In ieder geval, de metingen die men uitvoert op een evolutieproces zullen afhankelijk zijn van de tijd t .

Veel echte netwerkstructuren in de wereld hebben de zogenaamde *small world* eigenschap. Dit wil zeggen dat de meeste knopen kunnen bereikt worden vanuit andere knopen door een pad te volgen met een klein aantal bogen. Deze eigenschap wordt bijvoorbeeld gevonden in sociale netwerken, waar iedereen in de wereld kan bereikt worden door een korte ketting van kennissen.

Complexe netwerken liggen in het algemeen in een abstracte ruimte, waar de positionering van de knopen geen betekenis heeft. Dit is bijvoorbeeld het geval voor proteïne-proteïne interactie-netwerken. Maar er zijn veel netwerken waar de positie van de knopen belangrijk is omdat die posities de netwerkevolutie beïnvloeden. Dit is bijvoorbeeld het geval bij netwerken die autowegen en Luchtverkeerverbindingen voorstellen, of zelfs Internetverbindingen op een kaart. Deze soort netwerken worden *geografische of spatiale netwerken* genoemd.

In geografische netwerken kan het bestaan van een directe verbinding tussen knopen van veel factoren afhangen, zoals bijvoorbeeld afstand, resources, territoriale beperkingen, Dit is uiteraard afhankelijk van de toepassing. Indien de knopen bijvoorbeeld de steden op een kaart voorstellen, dan kan het gewicht dat we plaatsen op elke boog bijvoorbeeld de fysische afstand op een autosnelweg zijn tussen de verbonden steden. De toepassing in dit hoofdstuk maakt gebruik van spatiale netwerken.

Men kan de metingen op een complex netwerk zo ingewikkeld maken als men zelf wilt. Maar voor dit hoofdstuk hebben we niet meer metingen nodig dan hierboven werd besproken. Voor een meer diepgaande besprekking van andere mogelijke eigenschappen en metingen verwijzen we door naar het survey (Costa, Rodrigues, Travieso, & Villas Boas, 2007).

9.2. Complexe netwerken om een shape te beschrijven

We zullen nu de toepassing van complexe netwerken bespreken zoals geformuleerd in (Ricardo Backes, 2009). Het kernidee is om een startnetwerk te laten evolueren en metingen van elke tussenliggende evolutievorm te verzamelen in één grote feature vector. De feature vector van een shape beschrijft dus evolutiekenmerken van het bijhorende complexe netwerk.

Opvallend in de besproken aanpak is echter dat de shape hier geen opeenvolging van punten moet zijn, maar gewoon een verzameling $S = \{s_1, s_2, \dots, s_N\}$ van N punten in het vlak. Elk punt s_i is

van de vorm (x_i, y_i) met x_i de coördinaat op de X-as en y_i de coördinaat op de Y-as. De punten van de shape zijn niet expliciet onderling verbonden met lijnstukken. Het is de nabijheid van de punten in deze verzameling die kan duiden op een rand, maar niet een expliciete connectie van de punten in de vorm van een lijnstuk!

Definieer een geografisch netwerk $G = (V, E, \omega)$. De verzameling knopen is gelijk aan de verzameling van punten, dus $V = S$. Het netwerk is compleet: tussen elk paar knopen s_i en s_j wordt een boog gedefinieerd. Het gewicht op een boog is de Euclidische afstand tussen de verbonden knopen:

$$\omega(i, j) = d(s_i, s_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

De waarden van de coördinaten mogen in principe uit \mathbb{R} komen, maar indien zij zijn afgeleid van de pixel-coördinaten uit een afbeelding zijn zij beperkt tot \mathbb{N} . Belangrijk: de bogen zijn *ongericht*. Dit komt eigenlijk omdat we de euclidische afstand gebruiken als de gewichten, en de Euclidische afstand is duidelijk symmetrisch. Merk ook op dat alle gewichten positief zijn.

In deze constructie van een netwerk zijn alle knopen ook van hetzelfde type. Er worden geen attributen bijgehouden die iets vertellen over de kleur van het punt of iets dergelijks.

Zoals we in paragraaf 9.1.1 hebben gezien, kan het complexe netwerk weergegeven worden met zijn $N \times N$ weight matrix W . Elke cel $w_{i,j}$ bevat nu $d(s_i, s_j)$. We gaan al deze afstanden normaliseren zodat ze in het interval $[0,1]$ vallen:

$$W = \frac{W}{\max\{w_{i,j} \in W\}}$$

De uitdrukking $\max\{w_{i,j} \in W\}$ noemt men ook wel de *diameter* van de 2D dataset.

Zoals hierboven opgemerkt is het netwerk G compleet, waardoor alle knopen met elkaar verbonden zijn via een boog. Dit is een zogenaamd *regulier* netwerk, waar alle knopen dezelfde degree hebben. Uit een regulier netwerk kunnen we niet veel nuttige informatie halen over de kenmerken van de shape S . Daarom is het noodzakelijk om dit reguliere netwerk te transformeren naar een nuttiger complex netwerk waar wel relevante informatie kan uitgehaald worden. We zullen hiertoe een evolutie beschouwen van het netwerk G waarbij we starten zonder bogen en vervolgens steeds meer bogen zullen toevoegen. Het aantal knopen zal tijdens de evolutie *niet* veranderen.

Beschouw continue tijd $t \in [0,1]$. We laten de tijd t starten op een bepaalde starttijd $t_{ini} \in [0,1]$ en we verhogen de tijd telkens met stap-grootte $t_{inc} \in [0,1]$. Beschouw een sequentie van tijdstippen (t_l) met $l \in \mathbb{N}$:

$$\begin{aligned} t_1 &= t_{ini} \\ \forall l > 1: t_l &= t_{l-1} + t_{inc} \end{aligned}$$

We beschouwen ook een bepaalde eindtijd $t_{end} \in [0,1]$ en beschouwen enkel tijdstippen t_l zodat

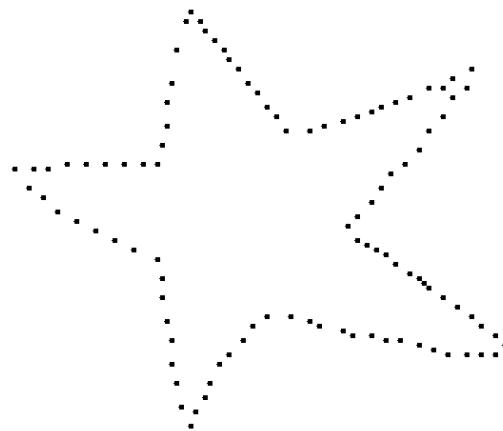
$$t_l \leq t_{end}$$

Stel dat er L zulke tijdstippen t_l bestaan. De tijdparameters t_{ini} , t_{inc} en t_{end} worden ingesteld door de gebruiker.

Hierboven hebben we gezien dat alle afstanden $w_{i,j}$ in weight matrix W genormaliseerd zijn binnen het interval $[0,1]$. Beschouw de thresholding van matrix W op tijdstip t_l ($1 \leq l \leq L$) door deze tijd zelf als threshold te gebruiken. Het resultaat is een adjacency matrix A^l :

$$\forall w_{i,j} \in W : \begin{cases} a_{i,j}^l = 0 & \text{als } |w_{i,j}| > t_l \\ a_{i,j}^l = 1 & \text{als } |w_{i,j}| \leq t_l \end{cases}$$

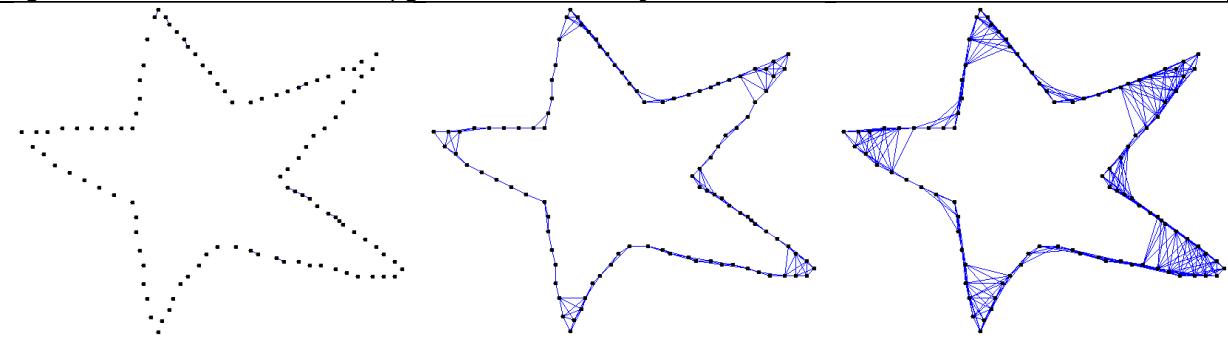
Beschouw de verzameling punten in Figuur 50. In Animatie 1 kan men de evolutie zien van het bijhorende complexe netwerk. De blauwe lijntjes stellen de bogen voor die verschijnen in de adjacency matrix wanneer we de threshold verhogen.

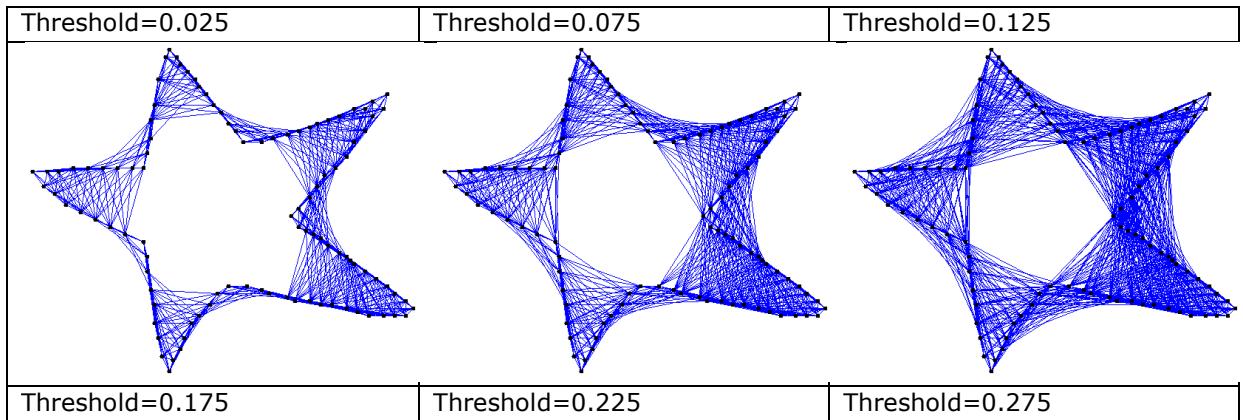


Figuur 50: een verzameling punten in het vlak, in de vorm van een ster. Dit voorbeeld is gebaseerd op (Ricardo Backes, 2009).

Animatie 1: evolutie van een voorbeeld complex netwerk

Deze illustratie van netwerkevolutie is gegenereerd met een eigen implementatie. Onder elke figuur is de threshold-waarde opgeschreven die erbij hoort.





De metingen die afgeleid worden van de groei van het complexe netwerk houden verband met de topologische eigenschappen van de shape. Zoals tevoren opgemerkt kunnen deze metingen gebruikt worden om een descriptor (in de vorm van een feature vector) af te leiden. De auteurs in (Ricardo Backes, 2009) beschouwen twee soorten descriptors, waarvan de eerste gebaseerd is op de degrees van de knopen en een tweede op een waarschijnlijkheid dat twee knopen met dezelfde degree met elkaar verbonden zijn. We zullen ons beperken tot het bespreken van deze eerste descriptor omdat deze tweede descriptor eigenlijk niet goed wordt uitgelegd en omdat deze eerste descriptor de beste experimentele resultaten heeft opgeleverd volgens de auteurs van het geraadpleegde werk.

Deze eerste descriptor heet de *degree descriptor*. Zoals zijn naam aangeeft zal de degree van elke knoop hierin een belangrijke rol spelen. Beschouw een vaste adjacency matrix A^l na thresholding op tijdstip t_l . Bereken voor elke knoop i ($1 \leq i \leq N$) de *genormaliseerde degree* k'_i . Dit is de degree van die knoop gedeeld door het aantal knopen N :

$$k'_i = \frac{k_i}{N}$$

Bereken nu de (genormaliseerde) average degree en maximum degree gebaseerd op alle k'_i :

$$k'_\mu(A^l) = \frac{1}{N} \sum_{i=1}^N k'_i$$

$$k'_{\max}(A^l) = \max \{k'_i | 1 \leq i \leq N\}$$

De degree descriptor φ is het samenvoegen van de genormaliseerde average en maximum degrees, over alle tijdstippen t_l ($1 \leq l \leq L$) heen:

$$\varphi = [k'_\mu(A^1), k'_{\max}(A^1), k'_\mu(A^2), k'_{\max}(A^2), \dots, k'_\mu(A^L), k'_{\max}(A^L)]$$

De descriptor φ is een array van grootte $2L$: er zijn L entries met average degrees en L entries met maximum degrees. De precieze volgorde van deze entries in de descriptor is niet belangrijk, zolang elke descriptor maar dezelfde volgorde gebruikt.

Door de boven genoemde normalisatie wordt de invloed van de netwerkgrootte op de descriptor gereduceerd. We zullen nu enkele eigenschappen van deze descriptor bespreken.

We noemen een functie $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ een *similarity transform* wanneer f de verhoudingen van de afstanden tussen punten behoudt. Formeel: $\exists c > 0, \forall p, q \in \mathbb{R}^2 : d(f(p), f(q)) = c \cdot d(p, q)$.

Dit is dus een bijzondere soort van topologische transformatie. Voorbeelden van similarity transforms zijn rotatie, *uniforme* schalering, translatie en spiegeling (t.o.v. een punt of een as). Een spiegeling kan natuurlijk uitgedrukt worden als een uniforme schalering, met eventueel een translatie indien niet ten opzichte van de oorsprong gespiegeld wordt.

- De descriptor φ is invariant onder eender welke similarity transform. We zullen dit intuïtief bespreken. Een uniforme schalering zal niets veranderen aan descriptor veroorzaakt omdat we de matrix W hebben genormaliseerd. De schalerings-invariantie is echter wel enkel bij uniforme schaleringen, waarbij eenzelfde schaleringsfactor wordt toegepast op alle coördinaatassen. Indien we dezelfde shape zouden beschouwen op verschillende grootten, zal door de deling van alle afstanden door de grootste afstand $d(s_i, s_j)$ dit geen invloed hebben op de descriptor omdat de grootste afstand ook op een uniforme manier mee geschaleerd wordt.
Indien we een rotatie toepassen, dan zullen de onderlinge afstanden tussen de punten bewaard blijven. De precieze coördinaten van de punten in het vlak speelt dus geen rol. Dit is ook meteen de reden waarom een translatie op de punten ook de descriptor niet zal veranderen.
Maar opgelet: stel dat we een shape baseren op pixel-afbeeldingen. Elke zwarte pixel kan tot de verzameling S gerekend worden. Dit is geen ondenkbaar scenario aangezien veel datasets enkel in pixel-vorm bestaan. Indien we deze pixel-shape bijvoorbeeld kleiner maken, dan zullen er minder pixels kunnen gebruikt worden om deze shape weer te geven. Ook de precieze ligging van de pixels in de gehele shape zal dan typisch lichtelijk variëren, door afrondingsfouten bij het omzetten van data in een groter grid naar data in een kleiner grid. Daardoor zullen de genormaliseerde Euclidische afstanden, berekend met deze nieuwe pixel-coördinaten, ook lichtelijk verschillen t.o.v. de genormaliseerde Euclidische afstanden in de oorspronkelijke grotere afbeelding. Bijgevolg gelden bovengenoemde invarianties in dit geval niet exact, maar benaderend. Dit zal meestal slechts kleine verschillen opleveren in de entries van de descriptor.
Door deze opmerking komen we meteen ook uit op de reden van de normalisatie van de degrees zodat enkel de degrees k'_i gebruikt worden om de average degree en maximum degree uit te rekenen per adjacency matrix A' . Twee sterk op elkaar gelijkende shapes S_A en S_B kunnen andere hoeveelheden punten bevatten. Dit geeft natuurlijk aanleiding tot verschillende complexe netwerken, dus netwerken met een verschillend aantal knopen en bogen. Dit geeft op zich aanleiding tot verschillende absolute degrees k_i . We willen echter dat de descriptors van beide omtrekken zeer dicht bij elkaar liggen en daarom worden de degrees genormaliseerd. Een absolute degree van 10 betekent iets heel anders in een klein netwerk dan in een groot netwerk. In het kleine netwerk duidt dit misschien op een grote connectiviteit van de knopen en in het grote netwerk op een lage connectiviteit. Door de absolute degrees te delen door het aantal knopen worden de degrees van beide netwerken opeens op eenzelfde schaalniveau gebracht waardoor ze wel zinvol kunnen vergeleken worden.
- Noise tolerantie: wanneer een omtrek wordt bekomen uit een afbeelding of via een of ander scanning-proces, kunnen kleine variaties en (afrondings-) fouten opduiken in de bekomen set van punten. Het is bijvoorbeeld mogelijk dat er "foutieve" punten in de verzameling zitten die helemaal niet tot de omtrek behoren. Deze kunnen redelijk ver van de echte omtrek afliegen, maar ook niet te ver want anders kan men ze eenvoudig(er) detecteren en elimineren op voorhand. De descriptor φ is gebaseerd op een

evolutieproces waarbij de bogen met de grootste Euclidische afstanden pas heel op het einde worden toegevoegd. De foutieve punten die een grotere afstand tot de echte omtrek hebben worden mogelijks pas aan het einde van een de evolutie toegevoegd. Indien we dan de descriptors vergelijken van een omtrek zonder noise en dezelfde omtrek met noise, dan zullen toch een groot aantal entries aan het begin van beide descriptors goed overeenkomen.

Bovendien is de helft van de entries in de descriptor gebaseerd op de genormaliseerde average degree. Door het nemen van gemiddelden worden de effecten van noise-punten op de descriptor ook al ingeperkt.

- Robuustheid. We hebben al verteld dat er geen expliciete lijnstukken aanwezig zijn tussen de punten. Hierdoor is ook geen volgorde nodig op de verzameling punten van de shape. Het is daarom toegelaten dat er gaten aanwezig zijn in de beschouwde shapes. We kunnen dus betekenisvolle descriptors afleiden die een omtrek van gebrekkige informatie karakteriseert.
In (Ricardo Backes, 2009) is de descriptor bijvoorbeeld gebruikt om verschillende klassen van bladeren te onderscheiden, gebaseerd op shapes waar sommige informatie van ontbreekt.

Wanneer eenmaal de descriptor is bekomen, is het de bedoeling dat de descriptors van meerdere shapes met elkaar worden vergeleken om vast te stellen of zij gelijkaardig zijn of niet. Het is verplicht dat elke descriptor precies dezelfde set van tijdstippen gebruikt heeft om de thresholding uit te voeren. De gebruiker stelt dus de tijdsparameters t_{ini} , t_{inc} en t_{end} in voor alle te beschouwen omtrekken (een hele database)!

9.3. Bespreking

Nog eens ter herhaling: het graafmodel beschouwt enkel de afstand tussen de shape-punten. Deze aanpak is dus meer robuust omdat het ook descriptors kan afleiden van shapes waar gaten inzitten. Sommige andere technieken voor het maken van descriptors voor omtrekken zijn sterk afhankelijk van het feit dat de omtrek gesloten is (Ricardo Backes, 2009). Bovendien is besproken aanpak invariant voor translatie, schalering en rotatie. Dit zijn zeer mooie eigenschappen.

De besproken aanpak is zowel *lokaal* als *globaal* van aard. In de eerste fazen van de netwerkevolutie zijn er enkel bogen tussen punten die dicht bij elkaar liggen. De eigenschappen van deze typisch erg sparse netwerken zeggen iets over zeer lokale eigenschappen van de shape. Deze informatie zit in de eerste entries van de descriptor φ . In latere fazen van de evolutie worden dan meer globale eigenschappen van de shape zichtbaar omdat punten over grotere afstanden verbonden worden door bogen. Deze informatie wordt samengevat in de laatste entries van de descriptor.

De auteurs van het geraadpleegde werk hebben hun techniek vooral toegepast op classificatie van shapes waarbij voor een query-shape een voorgedefinieerde klasse moet bepaald worden. Alle shapes in dezelfde klasse hebben een gelijkaardige visuele vorm. Het maken van de descriptor komt feitelijk overeen met het maken van een projectie naar punten in een meerdimensionale ruimte. Men kan in principe ook clustering uitvoeren in deze ruimte: groepeer de descriptor-punten die het dichtst bij elkaar liggen, bijvoorbeeld gebaseerd op de Euclidische afstand. Een descriptor wordt als *goed* bestempeld indien hij aanleiding geeft tot compacte clusters die redelijk ver van elkaar liggen.

Experimenten en conclusies die gebaseerd zijn op een eigen implementatie van de paper (Ricardo Backes, 2009) worden hieronder geformuleerd.

In het begin van dit hoofdstuk werd gezegd dat we een descriptor zouden maken voor shapes.

Maar het algoritme dat descriptor φ maakt kan werken met eender welke puntenwolk S als input. Dit komt omdat door de omzetting naar een complex netwerk volgorden op de punten geen rol spelen. Het enige wat belangrijk is zijn de punten zelf en hun onderlinge afstanden. We moeten ons dus niet beperken tot shapes alleen, maar kunnen ook punten beschouwen die niet per se in een visuele omtrek geschikt zijn. We zullen dit soort verzamelingen van punten een *patroon* noemen.

We zijn in deze thesis vooral geïnteresseerd in similarity searching. Een vorm van similarity searching is *range searching* waarbij voor een query-object alle objecten in een database worden gezocht die een afstand tot het query-object hebben die onder een drempelwaarde ε valt. In de context van dit hoofdstuk is de query een patroon dat door de gebruiker wordt ingevoerd. Voor de eenvoud is in de eigen experimenten geen ε -range gespecificeerd, maar werden alle patronen in de database eenvoudig gerangschikt volgens toenemende afstand van hun descriptor tot de descriptor van de query. De gebruikte afstand tussen twee descriptors φ_A en φ_B is in de eigen implementatie eenvoudigweg de Euclidische:

$$d(\varphi_A, \varphi_B) = \sqrt{\sum_{i=1}^{2L} (\varphi_A[i] - \varphi_B[i])^2}$$

In de eigen experimenten bestond de database uit zes verschillende patronen.

Beschouw de query in Figuur 51. De bijhorende database werd gesorteerd volgens dalende similarity. Het resultaat is getoond in ranking 1. Over deze ranking kunnen we tevreden zijn, aangezien ze vrij goed overeenkomt met onze visuele intuïtie. De laatste twee patronen zijn duidelijk verschillend en het algoritme zal hiervoor ook een descriptor maken die sterker verschilt. De eerste vier patronen lijken allemaal wel op elkaar, alleen zijn hier en daar wat details verschillend.

Het nadeel van descriptors in het algemeen is dat zij niet intuïtief interpreteerbaar zijn. Wanneer de afstand tussen twee descriptors klein is kunnen we niet meer in de originele puntenwolken aangeven waar deze schijnbare similarity zich situeert. Maar dat zou juist heel erg interessant zijn! Indien twee descriptors dicht bij elkaar liggen weten we nog altijd niet of zij ook visueel gelijkaardig zijn. Hiervoor hebben we een experiment uitgevoerd. In Figuur 52 ziet men het query-patroon. In ranking 2 kan men de resulterende rangschikking zien van de database. Het algoritme heeft dus de patronen die via hun descriptor het meeste "lijken" op de query vooraan geplaatst in de ranking. We zien dat de ranking niet overeen komt met onze visuele intuïtie. De nummers 3 en 5 zouden meer vanvoren hebben moeten staan. Hieruit blijkt duidelijk de beperking van de besproken techniek: we kunnen de descriptor niet gebruiken om voor een query-patroon alle patronen in de database te zoeken die elastische vervormingen zijn van deze query. Zoals verwacht scoort het besproken descriptor-algoritme zeer goed wanneer we patronen proberen terug te vinden die geroteerde, uniform geschaleerde en getransleerde versies zijn van de query. Maar sommige vormen die wij als mensen meer gelijkaardig vinden worden toch door het algoritme soms als minder gelijkaardig bestempeld. We besluiten dat descriptors eigenlijk niet zo krachtig zijn wanneer we patronen zouden willen terugvinden in een database die elastische deformaties zijn van de query. Deze elastische deformaties, zoals bijvoorbeeld *shearing*, zijn geen similarity transforms, omdat zij de verhoudingen van onderlinge afstanden van de punten niet respecteren: sommige punten worden door de deformatie dichter bij elkaar gebracht en andere verder uit elkaar.

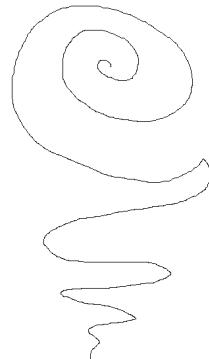
Het is goed mogelijk dat er een intelligentere afstandsfunctie tussen twee descriptors zou kunnen gedefinieerd worden die betere resultaten oplevert. Dit kan eventueel gebeuren door verschillende gewichten te associëren met de entries van de descriptor. Hier zullen we niet op ingaan omdat we eigenlijk vooral willen *zien* waar twee patronen overeenkomen. Met descriptors is dit zeer moeilijk

en daarom bestuderen we andere technieken die dit wél toelaten in een later hoofdstuk. We hebben in dit hoofdstuk echter geleerd dat grafen een krachtige techniek kunnen zijn om informatie weer te geven omtrent punten in het vlak en hun onderlinge relaties.

Er kunnen ook spatial access methods (SAMs) gebruikt worden om efficiënte range searching uit te voeren in de meerdimensionale projectieruimte waarin de descriptors gedefinieerd zijn. Een voorbeeld-SAM is de R-tree. Voor een introductie tot het gebruik van R-trees in de context van range searching op tijdreeksen kan men de thesis (Kellens, 2006) raadplegen. Wanneer eenmaal alle descriptors berekend zijn kan men via SAMs zeer efficiënt queries uitvoeren.

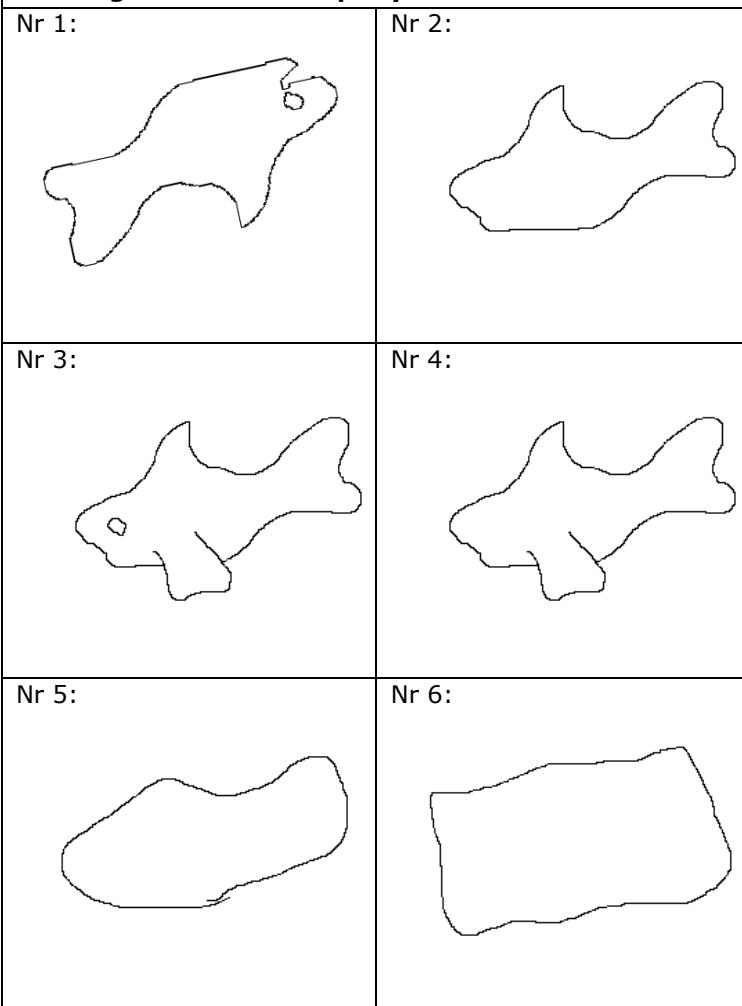


Figuur 51: query-patroon, de vorm van een vis

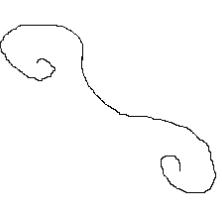
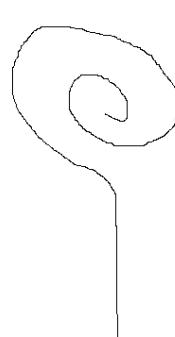
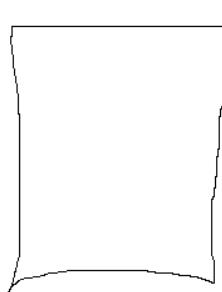


Figuur 52: query patroon, een spiraalachtige lijn

Ranking 1: voor de vis-query



Ranking 2: voor de spiraal-query

Nr 1:	Nr 2:
	
Nr 3:	Nr 4:
	
Nr 5:	Nr 6:
	

10. Association Graphs

We komen kort terug op het gebruik van wavelets bij Image Querying, zoals besproken in hoofdstuk 7. Het nadeel van wavelets is dat de wavelet decompositie zeer sterk afhankelijk is van hoe het signaal in het pixel-raster is opgeslagen. Wij kunnen als mensen makkelijk overeenkomsten ontdekken tussen twee verschillende afbeeldingen van eenzelfde voorwerp. Maar de wavelet-coëfficiënten kunnen drastisch verschillen tussen twee lichtelijk variërende afbeeldingen waardoor wavelets niet echt een flexibele methode zijn om verbanden te ontdekken tussen twee afbeeldingen. Proberen via wavelet-coëfficiënten te ontdekken of de ene afbeelding een elastische deformatie is van een andere lijkt dus zeer moeilijk of zelfs onhaalbaar.

Het is alom geweten dat grafen een krachtig uitdrukkingsmiddel zijn bij vele problemen in de informatica, getuige ook de toepassing in vorig hoofdstuk. Het is daarom niet ondenkbaar dat zij een belangrijke rol kunnen spelen om zeer complexe visuele gegevens te modelleren. In dit hoofdstuk wordt een algemene theorie besproken om similarities tussen verzamelingen van visuele objecten, in het vlak of in 3D, te ontdekken. Het is niet echt een veralgemening van vorig hoofdstuk, maar gewoon een andere manier waarop grafen kunnen gebruikt worden. Deze theorie opent de weg naar zeer interessante toepassingen van grafen in Pattern Recognition en Computer Vision. Een mooie survey-tekst waar ook andere technieken in besproken worden is (Conte, Foggia, Sansone, & Vento, 2007).

10.1. Inleiding

Deze inleidende paragraaf is gebaseerd op (Conte, Foggia, Sansone, & Vento, 2007).

Wanneer we een bepaald visueel object beschouwen als een knoop in een graaf kunnen we allerlei eigenschappen van een object opslaan als attributen in zijn bijhorende knoop. De bogen tussen deze knopen stellen bepaalde verbanden voor tussen de objecten. In het vorige hoofdstuk waren de visuele objecten de punten uit een shape en het verband tussen een paar punten was zéér eenvoudig, namelijk de Euclidische afstand.

Er zijn vele verschillende gebruiken van grafen mogelijk in deze context, afhankelijk van de volgende factoren:

- Welke visuele elementen wil men beschrijven?
- Hoe worden deze elementen gemapt naar knopen in een graaf?
- Welke attributen moeten we in een knoop bijhouden?
- Welke verbanden wil men beschrijven tussen de elementen die in de knopen zitten?
- Zijn de bogen gericht of ongericht?
- Welke attributen moeten we per boog bijhouden?
- Welke waarden nemen de attributen aan (numerisch, symbolisch, waarschijnlijkheden, ...)?

Het geheel van keuzen dat in een bepaald domein gemaakt wordt om een graaf op te stellen om visuele informatie te modelleren noemt men een *graafvoorstelling*.

Bijvoorbeeld, om de visuele inhoud van een afbeelding te beschrijven kan men eerst alle gebieden in deze afbeelding ontdekken waar een dominante kleur aanwezig is. Deze gebieden kan men in de graafvoorstelling laten overeenkomen met knopen die bijvoorbeeld een attribuut hebben waar de veelhoekcoördinaten van deze gebieden inzitten. Een boog tussen twee dergelijke knopen kan uitdrukken dat twee gebieden elkaar raken of in elkaar bevatten zijn. Hiervoor kan men bijvoorbeeld de ruwe topologische relaties uit paragraaf 8.1.1 gebruiken en deze aanvullen met concrete getalwaarden die kunnen zeggen hoe lang een gedeelde polyline-grens tussen twee veelhoeken is.

In Pattern Recognition en Computer Vision is het vaak noodzakelijk om visuele informatie te vergelijken en op zoek te gaan naar overeenkomsten. De toepassingen hiervan zijn eindeloos,

bijvoorbeeld: het herkennen van letters in een ingescand (en soms slecht leesbaar) document, zoeken naar gezichten van mensen in foto's, vingerafdrukken herkennen, Hiertoe worden de bijhorende grafen van al deze visuele informatie met elkaar "vergeleken". Stel dat er voor ons afbeeldingenvoorbeeld twee verschillende afbeeldingen beiden afzonderlijk zijn omgezet naar een graaf op de aangegeven manier (met gekleurde gebieden). Het is zeer waarschijnlijk dat door een verschillende visuele inhoud van beide afbeeldingen beide grafen een andere hoeveelheid knopen hebben en ook andere bogen.

De uitdaging is om voor elke graafvoorstelling een methode uit te werken waarmee zijn instanties kunnen vergeleken worden. Dit vergelijkingsproces noemt men *graph matching*. In (Conte, Foggia, Sansone, & Vento, 2007) worden zeer veel algoritmen uit de literatuur besproken om dit te doen. Er zijn twee grote hoofdcategorieën: *exacte methoden* en *niet-exacte (benaderende)* methoden.

10.1.1. Exakte methoden

Zij $G = (V, E)$ en $G' = (V', E')$ twee (gerichte of ongerichte) grafen. We vernoemen niet afzonderlijk de labelling-functies die labels plakken op de knopen en bogen, deze worden verondersteld. Ook veronderstellen we dat alle attribuutinformatie van een knoop of een boog in zijn label zit opgeslagen.

Bij exacte methoden moeten de labels van de gematchte knopen overeenkomen. Exacte methoden eisen ook dat wanneer twee knopen in de eerste graaf G verbonden zijn met een boog, hun beelden in de tweede graaf G' ook verbonden zijn door een boog die bovendien hetzelfde label draagt. De matching is dan *edge-preserving*.

Een conceptueel eenvoudige vorm van graaf-matching is *graph isomorphism* (graaf-isomorfie). Hier wordt gezocht naar een bijectie tussen V en V' die edge-preserving is.

Een graaf $G'' = (V'', E'')$ is een *subgraph* (deelgraaf) van G wanneer $V'' \subseteq V$ en $E'' \subseteq E \cap (V'' \times V'')$.

Een graaf $G'' = (V'', E'')$ is een (*vertex-*) *induced subgraph* van G wanneer $V'' \subseteq V$ en $E'' = E \cap (V'' \times V'')$. Het verschil tussen een gewone subgraph en een induced subgraph is subtiel maar zeer belangrijk. Het verschil zit in de manier waarop de bogen E'' worden gedefinieerd. Het woord *induced* drukt uit dat door een selectie van knopen $V'' \subseteq V$ voor de subgraph men automatisch de bogen moet meenemen die tussen deze knopen bestaan.

Bij het subgraph isomorphism (deelgraaf-isomorfie) probleem probeert men een isomorfie te construeren tussen de gehele graaf G en een subgraph van G' . Het induced subgraph isomorphism probleem is analoog, alleen wordt daar gewerkt met een induced subgraph van G' .

Een andere manier van graaf-matching is het zoeken van zogenaamde *maximal common induced subgraphs* (MCIS-probleem). Een *common induced subgraph* is een induced subgraph van graaf G die isomorf is met een induced subgraph van G' . Een *maximal common induced subgraph* kan niet meer uitgebreid worden naar een grotere deelgraaf met dezelfde eigenschap, vandaar "maximal". Hierbij is het toegelaten dat beide grafen buiten deze gedeelde topologie nog extra knopen en bogen hebben, maar géén extra bogen in hun gematchte deelgrafen. Zie definitie 7. In Figuur 54 worden de twee maximal common induced subgraphs getoond van de grafen in Figuur 53. We kunnen het resultaat van beide maximal common induced subgraphs noteren als

paren van knopen die overeenkomen tussen G en G' : $\{(n_1, m_1), (n_2, m_2)\}$ en $\{(n_2, m_2), (n_3, m_3)\}$. Merk op dat de graaf in Figuur 55 volgens definitie 7 geen maximal common induced subgraph is omdat er in grafen G en G' een boog moet zijn tussen de knopen met labels a en c .

Het maximal common induced subgraph probleem is NP-compleet (Sutera, Abu-Khzam, Zhang, Symons, Samatova, & Langston, 2005).

Opmerking: het zoeken naar *maximal common subgraphs* is een apart probleem dat ook NP-compleet is (Koch, 2001).

Definitie 7: Maximal Common Induced Subgraph

Zij $G = (V, E)$ en $G' = (V', E')$ twee grafen. Een *common induced subgraph* van G en G' is een graaf G'' zodat er een induced subgraph isomorphism bestaat van G'' naar G en een induced subgraph isomorphism van G'' naar G' .

G'' wordt een *maximal common induced subgraph* van G en G' genoemd wanneer er geen andere common induced subgraph bestaat waar G'' een deel van is. Het is mogelijk dat er meerdere maximal common induced subgraphs voor twee grafen G en G' bestaan, die eventueel zelfs knopen (en dus bogen) kunnen delen.

De common induced subgraph voor de twee grafen G en G' met het meeste aantal knopen in wordt de *maximum common induced subgraph* genoemd.

Wegens zijn strikte vereisten wordt graaf-isomorfie niet echt veel gebruikt in Pattern Recognition omdat grafen die verschillende, lichtelijk variërende versies van een bepaald visueel patroon voorstellen toch vaak verschillen kunnen hebben die veroorzaakt zijn door noise of het weggevallen van informatie. Deelgraaf-isomorfie en het zoeken naar maximal common induced subgraphs kunnen wel gebruikt worden om een bepaald patroon terug te vinden in een groter geheel of om overeenkomsten te ontdekken tussen visuele objecten.

Definitie 8: maximale, niet-maximale en maximum clique

Zij $G = (V, E)$ een graaf. V is de verzameling knopen en E is de verzameling bogen. Een subgraph $G' = (V', E')$ van G is een *clique* indien G' compleet is, dus alle mogelijke bogen bevatten ($E' = V' \times V'$). Een clique is een *maximale clique* indien hij geen deelgraaf is van een andere clique van G . Een *maximum clique* is een maximale clique met de grootste hoeveelheid knopen erin. Het is mogelijk dat er in een graaf G meerdere maximale en maximum cliques zitten. Het is mogelijk dat verschillende maximale cliques knopen en bogen gemeenschappelijk hebben.

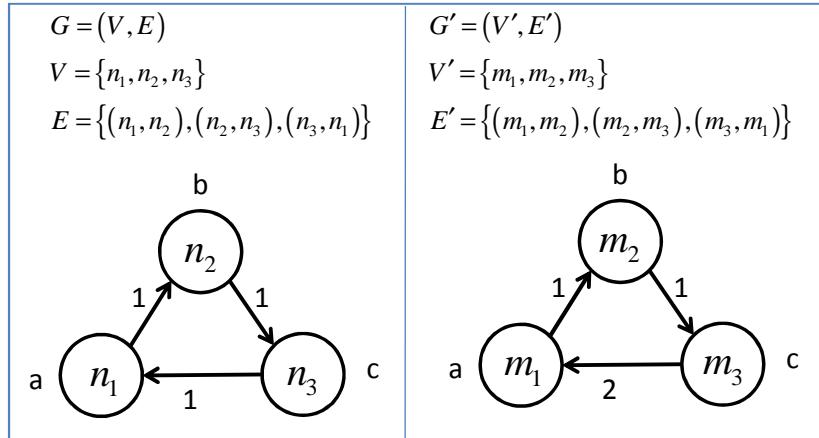
We noemen een clique *niet-maximaal* wanneer hij bevattet zit in een grotere clique.

Wanneer een clique n knopen bevat, dan noemen we die clique een n -clique.

Beschouw definitie 8. In (Levi, 1973) wordt opgemerkt dat het lastig is om een efficiënte heuristiek te bedenken voor het zoeken naar maximal common induced subgraphs voor twee grafen. Levi leidt af dat het MCIS-probleem echter kan omgezet worden naar het vinden van maximal cliques in een nieuwe soort graaf die aangeeft welke onderdelen van de twee grafen mogelijk kunnen overeenkomen. Deze nieuwe graaf wordt de *association graph* genoemd (Conte, Foggia, Sansone, & Vento, 2007), (Koch, 2001). Een association graph is een graaf die mogelijke overeenkomsten van knopen (en bogen) voorstelt tussen twee grafen. Dit wordt gedetailleerd uitgewerkt met een voorbeeldtoepassing in paragraaf 10.2. Het grote voordeel van de maximal clique aanpak is dat er redelijk snelle algoritmen voor bestaan, zie hoofdstuk 11.

Hoewel exacte graaf-matching in de worst case exponentieel veel tijd kan vergen is in veel toepassingen de rekentijd nog aanvaardbaar wegens volgende redenen:

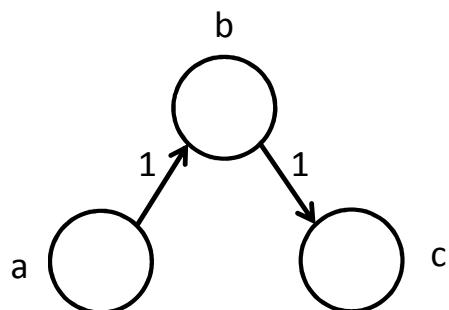
- De grafen hebben meestal niet de eigenschappen van het worst case geval.
- De attributen op de knopen en de bogen reduceren de rekentijd. Zoals we later zullen zien is dit mogelijk bij association graphs.



Figuur 53: twee voorbeeld-grafen G en G'

	subgraph of G	G''	subgraph of G'
MCIS 1			
MCIS 2			

Figuur 54: de twee maximale common induced subgraphs voor het grafen-paar van Figuur 53.



Figuur 55: dit is geen maximale common induced subgraph voor het grafen-paar van Figuur 53.

10.1.2. Niet-exakte methoden

Exacte graaf-matching methoden zijn soms nog iets te stroef om twee grafen te vergelijken. Zoals hierboven opgemerkt kan de topologie van de grafen verschillen doordat visuele patronen kunnen afwijken door noise of het wegvalLEN van data. Maar het is niet enkel de topologie die kan veranderen, maar ook de concrete waarden voor alle knoop- en boog-attributen. De matching methode moet kunnen afrekenen met deze verschillen door ook te versoepelen op de attributen (labels) van de knopen en bogen. We geven een voorbeeld, ook verdergaand op paragraaf 8.1.1. Stel dat we opnieuw knopen laten overeenkomen met veelhoeken en de binaire relaties tussen veelhoeken plaatsen als attribuut op de bogen. In paragraaf 8.1.1 hebben we gesproken over de *conceptual neighborhood* en welke versoepeling hiermee mogelijk wordt. Concreet kunnen we als versoepeling bijvoorbeeld toelaten dat een knopen-paar (a,b) in de eerste graaf gematcht wordt met een knopen-paar (c,d) in de tweede graaf waarbij het label van boog $a \rightarrow b$ "disjoint" is en het label van boog $c \rightarrow d$ "meet". We moeten bij niet-exakte methoden geen rekening houden met de edge-preserving eigenschap. Ook de labels van de gematchte knopen mogen zelfs van elkaar verschillen. Bijvoorbeeld, bij numerieke knoop-attributen mogen knopen bijvoorbeeld nog gematcht worden als het verschil van hun attributen onder een bepaalde threshold valt.

Er wordt bij niet-exakte methoden een *cost* berekend die uitdrukt hoe slecht (de waarden van) de attributen van de gematchte knopen en bogen op elkaar lijken. De hele graaf-matching probeert dan deze waarde dan te minimaliseren. Bovendien kan ook een kost in rekening gebracht worden als er tussen gematchte knopen een boog is in de eerste graaf maar niet in de tweede graaf.

Optimale niet-exakte algoritmen vinden altijd een globaal minimum voor deze kost indien dit bestaat. Suboptimale niet-exakte matching algoritmen vinden gewoon een lokaal minimum, misschien niet al te ver van het globale minimum. De association graph methode vindt een globaal optimum.

10.2. Spatial Scene Similarity

In deze paragraaf wordt via een concrete toepassing het concept van association graphs geïntroduceerd en geïllustreerd, namelijk het queryen van GIS-data. Het is trouwens via deze toepassing dat we voor het eerst deze techniek vernamen. Deze paragraaf is sterk gebaseerd op (Nedas & Egenhofer, Spatial-scene Similarity Queries, 2008) en in mindere mate op (Egenhofer, 1997).

We zullen de theorie bespreken om similarity queries uit te voeren op "configuraties" van 2D objecten in het vlak. De objecten die wij beschouwen kunnen punten, lijnstukken, veelhoeken, ... zijn. Een configuratie is een verzameling van 2D objecten die op een bepaalde manier gepositioneerd zijn ten opzichte van elkaar, zoals bijvoorbeeld gebouwen die bij elkaar staan op een landkaart. De gebruiker stelt een query-configuratie op waarin hij een aantal objecten heeft getekend. Vervolgens kan hij deze configuratie aanbieden aan een database. Er zijn twee mogelijke scenario's:

- 1) de database bevat meerdere individuele records die aparte configuraties van objecten voorstellen
- 2) de database is één continue configuratie van 2D objecten. Een landkaart is een mooi voorbeeld hiervan.

Het eerste scenario lijkt op de voorgaande hoofdstukken waar de database individuele afbeeldingen of patronen bevatte. Daaruit moesten we dan de records uitzieken die het meest gelijken op de query. In het tweede scenario is het de bedoeling dat de gebruiker met zijn query een kleiner stukje uitdrukt dat we (eventueel meermaals) moeten terugvinden in de hele "kaart" van de database.

Stel dat een gebruiker in een query wil beschrijven dat een rivier doorheen een perceel loopt en dat aan één kant van deze rivier een bos gelegen is. Het is de taak van de database om alle plaatsen in een kaart of een serie van kaarten te vinden die aan deze setting voldoen. Het is niet meteen eenvoudig om deze query in SQL te verwoorden. Het zou veel eenvoudiger zijn een soort (abstracte) tekening te maken van deze configuratie die toch door de database kan geïnterpreteerd worden. Spatial-query-by-sketch is een alternatief voor tekstuele SQL-queries op spatial databases die toelaat de query weer te geven als tekening waarin een aantal 2D objecten op een bepaalde manier getekend zijn ten opzichte van elkaar. In (Egenhofer, 1997) wordt nog opgemerkt dat sommige constraints op de tekening zoals precieze maten van afstand of bepaalde attributen van objecten eventueel beter met tekstuele constraints kunnen verwoord worden. Maar daar gaan we momenteel niet dieper op in.

De gebruiker tekent enkel een benaderende versie van wat hij wil terugvinden. De besproken theorie neemt aan dat het mogelijk is om een query-configuratie te matchen met een bepaald stuk uit de database indien een aantal meest belangrijke criteria overeenkomen tussen de query en dit stuk. Het doel van deze paragraaf is om aan te geven wat voor soort criteria interessant kunnen zijn. We zullen voor de eenvoud even doen alsof er gewoon slechts twee configuraties Q en D bestaan. Hierbij is Q de query en D een configuratie uit de database en zij duiden beiden gewoon een verzameling objecten aan in het vlak. Het is de bedoeling dat een gedeeltelijke mapping $m: Q \rightarrow D$ gemaakt wordt die objecten $a \in Q$ afbeeldt op objecten $x \in D$. We zullen voor de eenvoud veronderstellen dat m injectief is. Dan heeft elk object uit de query een uniek beeld. Laten we ook veronderstellen dat elk query-object hoogstens één beeld heeft. Het is dus niet vereist dat alle objecten uit de query-configuratie een overeenkomend object in de database-configuratie hebben of omgekeerd.

Bitmap data kan wel een goede vorm zijn om een schets in te representeren, maar het is moeilijk om de gegevens erin te interpreteren en de schijnbare vormen erin te matchen met elementen in andere configuraties in een database. Een object-representatie laat toe om details weg te abstracteren en enkel te focussen op de hoofdvormen van een configuratie. De data waar in deze paragraaf mee gewerkt wordt is vectordata in de vorm van polylines en polygons. Al deze objecten kunnen opgeslagen zijn in een topologisch vector datamodel zodat bijvoorbeeld geweten is of een bepaald object overlapt met een ander of in een ander object bevindt of niet. Het topologisch datamodel is gekend uit GIS, meer concreet in spatial databases. Deze object-representatie legt de nadruk op de objecten en hun onderlinge binaire relaties, die we hier *spatial relations* noemen.

10.2.1. Semantisch Netwerk en formulering matching probleem

Een ruimtelijke configuratie wordt voorgesteld als een graaf, het zogenaamde *semantisch netwerk*. Dit is dus een graafvoorstelling van de visuele informatie. Hierbij zijn de knopen gelijk aan de objecten. Een boog tussen knopen i en j komt overeen met de spatial relation tussen objecten i en j . Per knoop kunnen we geometrische- en attribuut-informatie bijhouden over het object: vorm, kleur, In geografische databases kan een object bijvoorbeeld een gebouw voorstellen. De bijhorende vorm is dan een veelhoek en er kan een extra attribuut worden bijgehouden om aan te geven dat het object van type "gebouw" is. Het is toegelaten dat elk object dezelfde attributen heeft, voor zover dit een zinvolle betekenis heeft. Bij praktische implementaties is het in ieder geval nuttig als elk object een bepaald type-attribuut heeft zoals "lijn" of "veelhoek". Op die manier is het mogelijk lijnen uit de query-configuratie in verband te brengen met lijnen in de database-configuratie. Natuurlijk is het mogelijk dat elk 2D object van een bepaald type eigen extra attributen heeft. Een lijn kan bijvoorbeeld een lengte-attribuut lengte hebben en een veelhoek een oppervlakte-attribuut. Het is

ook bij de spatial relations tussen objecten toegelaten dat dezelfde relatie-attributen voor elk paar objecten gebruikt worden. Deze attributen zijn dan noodzakelijk meer algemeen.

In Figuur 56 wordt een voorbeeld-configuratie van 2D objecten in het vlak getoond. In het bijhorende semantische netwerk, getoond in Figuur 57, zijn alle mogelijke spatial relations tussen de objecten weergeven. Merk op dat relaties tussen objecten in het algemeen niet symmetrisch hoeven te zijn, daarom kan men het semantisch netwerk modelleren als een gerichte graaf. Elke boog van knoop i naar knoop j bevat alle informatie omtrent de asymmetrische relatie van object i naar object j . Indien men niet geïnteresseerd is in richtingsafhankelijke informatie kan de informatie op boog (i, j) gelijk gesteld worden aan de informatie op boog (j, i) , waardoor het semantisch netwerk een ongerichte graaf is.

Concrete voorbeelden van spatial relations zijn reeds besproken in paragraaf 8.1.1. Ter illustratie zullen we deze high-level topologische relaties schrijven op de bogen die vertrekken vanuit object A. Zie Figuur 58.

In definitie 9 geven we een formele definitie van een semantisch netwerk. Deze definitie is zeer algemeen en kan zelfs gebruikt worden om eigenschappen en binaire relaties van eender welke objecten voor te stellen, dus niet per se objecten in het vlak.

Definitie 9: Semantisch Netwerk

Semantisch netwerk G is een complete gelabelde, gerichte of ongerichte, graaf met volgende componenten:

- 1) V_G , een verzameling knopen. Elke knoop $v \in V_G$ stelt een object voor uit het *object-universum* U_{obj} .
- 2) $\lambda_G : V_G \cup (V_G \times V_G) \rightarrow U_{str}$. De functie λ_G is de labelling-functie die aan elke knoop uit V_G en boog uit $(V_G \times V_G)$ een label toekent. Alle labels komen uit het *label-universum* U_{str} .

We noteren $G = (V_G, \lambda_G)$.

Merk op dat we de verzameling bogen niet expliciet een naam hebben gegeven (tevoren deden we dat steeds met de letter E). Dit komt omdat graaf G compleet is en bijgevolg sowieso alle bogen aanwezig zijn. Deze informatie moet bijgevolg niet met een aparte verzameling gemodelleerd worden.

Een label $l \in U_{str}$ kan meerdere attributen bevatten, dit kan verschillen per label. Het definiëren en parsen van labels is zeer applicatie-specifiek. In een GIS-toepassing kan een label voor een object als attributen bijvoorbeeld de positie bijhouden, zijn kleur, We veronderstellen ook dat elke applicatie een afstandsfunctie $d : U_{str} \times U_{str} \rightarrow \mathbb{R}^+$ definieert tussen labels. Hiermee is het mogelijk om te berekenen hoe sterk labels met elkaar overeenkomen (door de attributen te vergelijken).

Tenslotte, afhankelijk van de applicatie is het niet uit te sluiten dat $U_{obj} \cap U_{str} \neq \emptyset$.

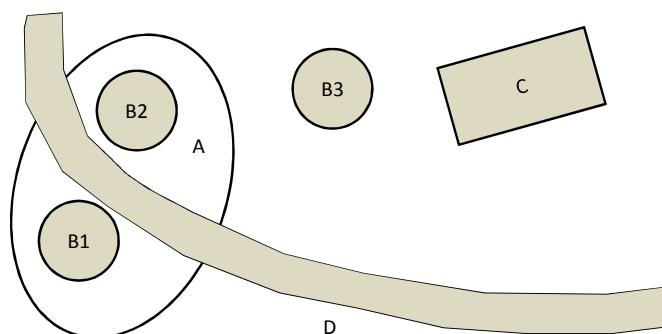
In paragraaf 10.2.1.1 is het eenvoudiger als er geen richtingsafhankelijkheid is bij de spatial relations tussen objecten. Het semantisch netwerk is dan een *ongerichte graaf*. Eventuele

richtingsafhankelijke informatie kan dan in gepaste attributen van de ongerichte bogen voorkomen als "subdata". We zullen vanaf nu *symmetrische* binaire relaties tussen objecten veronderstellen waar eventueel de richtingsafhankelijke informatie in verwerkt zit.

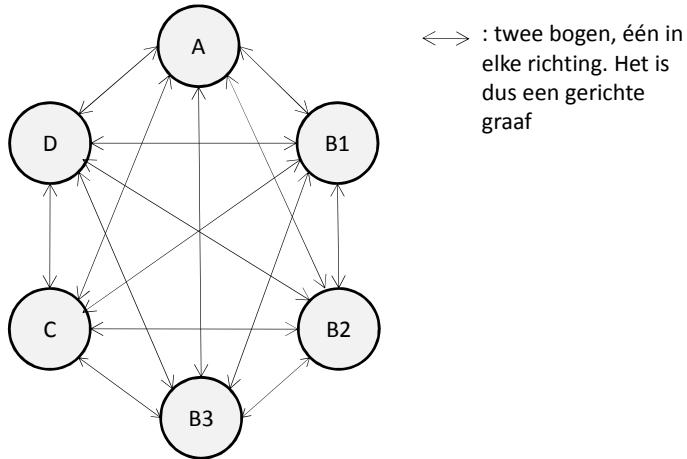
Beoordelen of twee ruimtelijke configuraties gelijkaardig zijn is niet eenvoudig omdat er cognitieve en computationele aspecten aan verbonden zijn. Het is belangrijk om vast te stellen welke objecten van de query-configuratie overeenkomen met welke objecten uit de database-configuraties zodat de verbanden in de bijhorende semantische netwerken ook redelijk goed overeenkomen. In (Nedas & Egenhofer, 2008) worden een aantal spatial matching principes gemotiveerd vanuit de cognitieve psychologie:

- 1) Mensen beginnen met het lokaliseren van mogelijke object-object matches tussen de twee configuraties. Heel verschillende objecten worden eerder genegeerd dan gedwongen met elkaar te matchen.
- 2) Mensen associëren objecten zodat de relaties tussen de objecten in de query-configuratie ook ongeveer behouden zijn tussen de overeenkomende objecten in de database-configuratie.
- 3) Wanneer er meerdere matchings mogelijk zijn, probeert men de globale "fit" met de query-configuratie te optimaliseren. Er wordt dus gekozen voor een aantal objecten uit de database-configuratie die het minste verschilt (op zoveel mogelijk aspecten) met de query-configuratie.

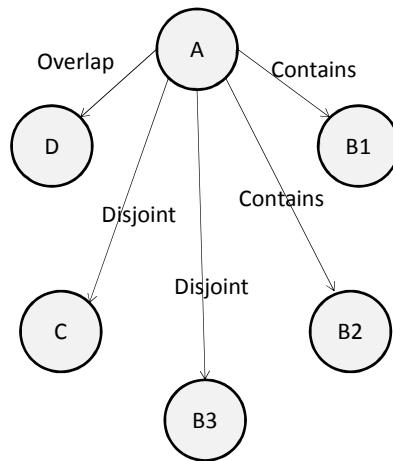
Er wordt door de auteurs voorgesteld om bovenstaande principes te gebruiken bij het matchen van het semantische netwerk van de query-configuratie met het semantische netwerk van de database-configuratie. Omdat we in een semantisch netwerk heel duidelijk de relaties tussen de objecten hebben gemodelleerd, zal het matchen zich niet alleen richten op attribuut-eigenschappen van individuele objecten, maar ook op de spatial relations tussen deze objecten. We gebruiken de semantische netwerken eigenlijk enkel als een conceptueel hulpmiddel en deze hoeven in principe niet expliciet te worden geconstrueerd in een praktische toepassing. Maar het vergelijken van twee ruimtelijke configuraties kan via de definitie van semantische netwerken voorgesteld worden als een graaf-matching probleem, waardoor het probleem conceptueel toegankelijker wordt.



Figuur 56: voorbeeld-configuratie



Figuur 57: voorbeeld semantisch netwerk dat hoort bij de objecten van Figuur 56.



Figuur 58: high-level topologische relaties geschreven op de bogen vanuit knoop A in het semantisch netwerk van Figuur 57.

Gegeven zijn nu twee semantische netwerken, eentje voor de query-configuratie en eentje voor de database-configuratie. Deze database-configuratie kan bijvoorbeeld één entry zijn uit een grote database of gewoon de hele database (een grote kaart) zijn. We zullen voor de eenvoud spreken over de query-graaf respectievelijk de database-graaf wanneer we willen verwijzen naar de semantische netwerken.

Concreet wordt in (Nedas & Egenhofer, 2008) voorgesteld om te zoeken naar maximal cliques in een zogenaamde association graph. In paragraaf 10.1.1 is gezegd dat hiermee het *maximal common induced subgraph* probleem iets efficiënter kan opgelost worden. Het is bovendien nuttig om het niet-exacte matching probleem hiervan te beschouwen. Immers, door de onnauwkeurige aard van sommige queries wordt het vinden van een exacte match soms zeer onwaarschijnlijk. Het is mogelijk dat de attributen van de objecten en hun relaties in de query niet exact matchen met de gegevens in de database. Wanneer de objecten bijvoorbeeld veelhoeken zijn, kan het zeker voorkomen dat we een veelhoek uit de query willen matchen met een veelhoek uit de database waarvan de vorm (eventueel lichtjes) verschilt. Vergelijkbare scenario's zijn ook denkbaar voor het niet-exact matchen van de binaire relaties. Daarom zal het nodig zijn om sommige (onnauwkeurige) gegevens in de query-graaf wat te versoepelen en geen exacte match te eisen.

Stel dat de query n objecten bevat en de database N . Wanneer we helemaal geen beperkingen voorzien van de query op de oplossingen (de ultieme versoepeling) is het aantal mogelijke oplossingen gelijk aan: $N \cdot (N-1) \cdot (N-2) \dots (N-n+1)$.

Hierbij veronderstellen we een vaste volgorde op de objecten in de query zodat we enkel een geschikte volgorde van de database objecten moeten vinden om eraan toe te kennen. Gezien bovenstaande uitdrukking is het zelfs al voor redelijk kleine databases en queries heel zwaar om gewoon alle mogelijke matchings uit te proberen tussen objecten uit de query en objecten uit de database. Daarom moeten we goed nadenken over hoe sterk we de originele gegevens in de query-graaf mogen versoepelen.

Veronderstel een *globale threshold* T die het grootst toelaatbare verschil tussen een query-configuratie en een gematchte database-configuratie uitdrukt. Stel dat een mogelijke kandidaat-matching is gevonden van query-objecten naar database-objecten. Er wordt op een bepaalde (applicatie-specifieke) manier een waarde berekend die uitdrukt hoe groot het verschil is van de database-objecten t.o.v. de query-objecten. Dit gebeurt op een globale manier, door bijvoorbeeld het gemiddelde te nemen van de waarden berekend uit goede object-object matches en de waarden uit slechte object-object matches. Bij globale thresholding wordt de hele kandidaat-matching aanvaard indien deze gemiddelde globale waarde onder de globale threshold T ligt. *Lokale thresholds* kunnen toegepast worden op individuele matchings van objecten of individuele relaties tussen gematchte objecten. Hierbij wordt gekeken of het verschil tussen twee attributen van gematchte knopen of bogen onder een bepaalde versoepelings-threshold ligt. Er wordt *per knoop* en *per boog* van de query een afzonderlijke lokale threshold gespecificeerd.

Het toepassen van een globale threshold komt overeen met een *soft retrieval* strategie waarbij de oplossingen "gemiddeld goed" zijn. Het is mogelijk dat er in een oplossing een deel van de database-objecten zeer goed of perfect matcht met de objecten en relaties uit de query, maar een ander deel van de database-objecten in de oplossing zeer slecht matcht. Het geven van dergelijke oplossingen kan dan de psychologische motivaties schenden van paragraaf 10.2.1, namelijk dat mensen mappings proberen te maken die zo goed mogelijk zijn en niet mappings waar een deel van de query-objecten heel goed gematcht wordt en een ander deel heel slecht. Bovendien heeft een soft retrieval strategie last van performantie-problemen omdat veel verschillende oplossingen moeten beschouwd worden en gedeeltelijk *geïnstantieerd* moeten worden om te beslissen of zij moeten verworpen worden of niet. Indien de globale threshold verhoogd wordt zal dit efficiëntieprobleem alleen maar toenemen!

Wanneer we enkel maar lokale thresholds toepassen, dan hebben we een soort van *hard retrieval* strategie. Een oplossing (volledig of onvolledig) moet in dat geval zo goed mogelijk presteren op elke lokale threshold, zowel voor knopen als voor bogen. Indien één van de lokale thresholds wordt overschreden, wordt de hele oplossing verworpen. Een hard retrieval strategie is eigenlijk sterk verkiesbaar boven een soft retrieval strategie wegens volgende redenen:

- 1) Één van de redenen is *controle*. We kunnen tijdens het versoepelen heel precies beslissen welke verschillen in waarden we *per attribuut* nog tolereren en welke niet. Indien we meer oplossingen willen toelaten, kunnen we eenvoudig begrijpen hoe we de lokale thresholds op de attributen moeten aanpassen zodat zij meer oplossingen doorlaten.
- 2) Een tweede reden is *psychologische motivatie*: juiste lokale thresholds benaderen de perceptie van een mens. De juiste lokale thresholds kunnen er immers voor zorgen dat er matchings van hoge kwaliteit zijn tussen objecten en relaties die *lokaal van aard* zijn in de hele configuratie.
- 3) Een derde reden is *optimalisatie*: we kunnen eenvoudiger indexatie-structuren bedenken voor object-object matches of relatie-relatie matches dan voor grotere groepen objecten tegelijk.

Het kiezen van lokale thresholds voor attributen is applicatie-specifiek omdat de attributen zelf applicatie-specifiek zijn. In paragraaf 8.1.1 hebben we de conceptual neighborhood graph gezien voor high-level topologische relaties tussen twee veelhoeken. Per veelhoek-veelhoek boog in een semantisch netwerk wordt dan een attribuut bijgehouden waar één van deze topologische relaties als waarde in voorkomt. Een lokale threshold zou nu bijvoorbeeld het getal "2" kunnen zijn dat uitdrukt dat twee relaties nog gelijkaardig zijn wanneer zij elkaar kunnen bereiken met twee sprongen in de conceptual neighborhood graph. Wanneer een relatie tussen twee veelhoeken in de database drie sprongen verwijderd is van de relatie tussen de veelhoeken in de query, dan is deze lokale threshold op dit attribuut overschreden en wordt de oplossing verworpen. Het had best gekund dat alle andere gematchte veelhoek-relaties in de oplossing wel onder de threshold bleven, maar toch wordt de hele oplossing verworpen! Dit voorbeeldje geeft mooi aan hoe intuïtief deze lokale thresholds zijn om te begrijpen en om mee te werken.

10.2.1.1. Concreet matching algoritme

Zij $G = (V_G, \lambda_G)$ het semantische netwerk van de query-configuratie en zij $H = (V_H, \lambda_H)$ het semantische netwerk van de database-configuratie. We veronderstellen voor de eenvoud dat zij ongericht zijn, zoals ook reeds tevoren opgemerkt. Zoals tevoren opgemerkt, gebruiken we de semantische netwerken enkel als een conceptueel hulpmiddel en deze hoeven in principe niet expliciet te worden geconstrueerd in een praktische toepassing.

We veronderstellen een aparte functie $\tau: V_G \cup (V_G \times V_G) \rightarrow \mathbb{R}$ die aan elke knoop en boog van G een lokale threshold toekent. We maken een speciale graaf, de *association graph* genoemd. De association graph $A = (V_A, E_A)$ is *ongericht*. De verzameling knopen V_A wordt als volgt gedefinieerd:

$$V_A = \{a_{u,v} = (u, v) \in V_G \times V_H \mid d(\lambda_G(u), \lambda_H(v)) \leq \tau(u)\}$$

Voor elk paar van "compatibele" knopen wordt dus een knoop in de association graph gemaakt. In een tweede faze worden de bogen van de association graph gegenereerd door knopen van de association graph te verbinden zodat de relatie tussen de betrokken database-objecten voldoende lijkt op de relatie tussen de overeenkomstige query-objecten. De verzameling bogen E_A wordt formeel als volgt gedefinieerd:

$$E_A = \{(u_1, v_1), (u_2, v_2) \} \in V_A \times V_A \mid u_1 \neq u_2 \wedge v_1 \neq v_2 \wedge d(\lambda_G(u_1, u_2), \lambda_H(v_1, v_2)) \leq \tau((u_1, u_2))\}$$

We verbieden momenteel bogen tussen knopen $a_{u1,v1}$ en $a_{u2,v2}$ wanneer $u_1 = u_2$ of $v_1 = v_2$. Dit drukt uit dat een query-object hooguit met één database-object kan matchen en een database-object ook hooguit met één query-object kan matchen. Dit zal zometeen duidelijk worden.

Ook al beginnen we bij het opstellen van de association graph met het matchen van compatibele objecten en pas daarna de relaties, betekent dit toch niet dat er een voorkeur uitgaat naar het doen overeenkomen van objecten ten opzichte van het laten overeenkomen van relaties tussen deze objecten. Men kan evengoed beginnen met het beschouwen van alle mogelijke relaties tussen objecten in G en dan voor elke query-relatie een relatie zoeken tussen objecten in de database die voldoet aan de lokale thresholds van die query-relatie. Dit geeft aanleiding tot dezelfde association graph.

Merk op dat het aantal knopen in de association graph ook afhangt van hoe goed de objecten zelf met elkaar matchen. Het maximaal aantal mogelijke knopen is $|V_G| \cdot |V_H|$. In dat geval kan gezegd elk query-object matchen met elk database-object.

Wanneer we eenmaal de association graph gemaakt hebben, gaan we erin op zoek naar maximum en andere maximale cliques. Het zoeken van een zo *volledig* mogelijke oplossing komt overeen met het zoeken naar een maximum clique in de association graph. Het is niet gegarandeerd dat de mapping van objecten die geïmpliceerd wordt door de maximum clique ook effectief een volledige oplossing is omdat het niet zeker is dat de maximum clique ook zoveel knopen bevat als er in de query aanwezig zijn. Alle gewone maximale cliques die niet maximum zijn komen sowieso overeen met onvolledige oplossingen omdat niet elk object in de query dan een bijhorend partner-object heeft in de database. Niet-maximale cliques zijn *redundante* oplossingen omdat zij deel uitmaken van een grotere oplossing.

Merk op dat door het beschouwen van cliques als oplossingen het onmogelijk is dat een query-object gematcht wordt naar meer dan één database-object. Stel even dat dit wel mogelijk zou zijn. Dan zitten er in een clique minstens twee association-knopen met hetzelfde query-object in. Maar in de constructie van de bogen van hierboven hadden we afgesproken dat er geen boog wordt toegevoegd tussen dergelijke knopen. Hierdoor kunnen beide knopen onmogelijk in dezelfde clique zitten! Via een gelijkaardige redenering kunnen we ook concluderen dat een database-object hoogstens met één query-object gematcht wordt.

Er is echter informatie die we nog aan de association graph moeten toevoegen om selectief goede oplossingen te kunnen vinden. We voegen aan elke boog een gewicht toe dat uitdrukt hoe goed de match tussen de relatie in de query en de relatie in de database feitelijk is. Hiertoe kan men per association-knoop $a_{u,v}$ een *dissimilarity score* berekenen. Hoe hoger dit getal, hoe minder goed de attributen van het query-object $u \in V_G$ overeenkomen met de attributen van het database-object $v \in V_H$. Dit is natuurlijk reeds in de veronderstelling dat aan de lokale threshold $\tau(u)$ is voldaan, want anders hadden we nooit knoop $a_{u,v}$ gemaakt. Indien het niet uitmaakt hoe goed een bepaald query-object matcht met het database-object, kan men gewoon een nul invullen als dissimilarity-waarde. Er staat ook een dissimilarity score op de boog tussen knopen a_{u_1,v_1} en a_{u_2,v_2} . Hoe hoger dit getal, hoe minder de relatie (u_1, u_2) in de query-configuratie lijkt op de relatie (v_1, v_2) in de database-configuratie. Indien het niet uitmaakt hoe een bepaalde relatie van de query matcht in de database, kan men gewoon een nul invullen als dissimilarity-waarde op de boog in de association graph.

In (Nedas & Egenhofer, 2008) wordt voorgesteld dat elke dissimilarity score berekend wordt door het aggregeren van dissimilarity scores die resulteren uit het vergelijken van meerdere attributen. In een relatie kunnen bijvoorbeeld kwalitatieve attributen zitten omtrent high-level topologische relaties of meerdere numerieke attributen. Men kan twee relaties dan eerst vergelijken door een score te geven per attribuut en daarna deze attribuut-scores samen te voegen tot één *macro-score* die iets zegt over de hele relatie in al zijn facetten. Op die manier kan elk attribuut een eigen gewicht krijgen toegekend en is het toekennen van scores ook eenvoudiger te begrijpen als een soort gewogen som van onafhankelijke factoren. De manier waarop dit gebeurt is natuurlijk ook weer applicatie-specifiek.

Eigenlijk kan men met de boven genoemde dissimilarity scores reeds een rangschikking maken van alle cliques: de beste oplossing is de grootste clique met de kleinste overall dissimilarity score. Maar het is misschien iets intuïtiever indien we zouden werken met *similarity scores* op de knopen en bogen in de association graph. Dan is de beste oplossing de grootste clique met de grootste overall similarity score. Hoe men deze overall score kan berekenen zien we zodadelijk. In (Nedas & Egenhofer, 2008) wordt kort vermeld dat dissimilarity scores kunnen omgezet worden naar

similarity scores via conversie-functies. Voor het gemak kan men deze functies zodanig maken dat de similarity scores liggen in het interval $[0,1]$. Het is vermoedelijk intuïtiever om algoritmen te implementeren die dissimilarity scores berekenen i.p.v. similarity scores. Maar omdat deze altijd automatisch kunnen omgezet worden naar similarity scores, zullen we vanaf nu enkel nog werken met similarity scores!

Hierboven is verteld dat indien het niet uitmaakt hoe sterk twee objecten of relaties overeenkomen men hiervoor een zeer kleine dissimilarity score kan invullen op de knopen respectievelijk bogen van de association graph, nul bijvoorbeeld.

Laten we dit dieper bespreken voor binaire relaties. Eigenlijk maakt de precieze dissimilarity-waarde die men invult op een boog in de association graph niet uit omdat er enkel met similarity scores zal gerekend worden. Het belangrijkste is gewoon dát men een boog trekt! Dit idee is eigenlijk zelf bedacht en niet gehaald uit de literatuur. De definitie van semantisch netwerk (definitie 9) vereist dat we iets kunnen zeggen over elke mogelijke binaire relatie tussen twee objecten, maar het kan voorkomen dat de binaire relatie tussen twee objecten A en B in de query eigenlijk niet zo belangrijk is. Bijgevolg maakt het niet uit hoe deze relatie wordt ingevuld door objecten in de database. Dit zou aanleiding moeten geven tot meer mogelijke oplossingen omdat die relatie dan geen extra beperking mag vormen. We kunnen daarom veronderstellen dat de lokale thresholds op deze niet-belangrijke relaties oneindig groot zijn. Hoe kunnen we echter *aangeven* dat een relatie niet belangrijk is in de association graph? Stel, we beschouwen alle objecten X in de database waar A mee matcht en alle objecten Y waar B mee matcht. De scores van de resulterende knopen in de association graph worden op de gewone manier berekend. Doordat de lokale threshold op de relatie tussen A en B oneindig hoog ligt, voegen we eigenlijk *onvoorwaardelijk* een boog toe tussen alle mogelijke knopen (A,X) en (B,Y). Deze bogen verhogen de kans op grotere cliques en kunnen zelfs eventueel door meerdere maximale cliques gebruikt worden. Omdat er gerekend zal worden met similarity scores, zullen we de similarity score op elke onvoorwaardelijke boog wél een concrete waarde moeten geven, namelijk nul. Alle verschillende maximale cliques die zo'n onvoorwaardelijke boog bevatten, hebben geen voordeel t.o.v. elkaar door de score op zo'n boog (want die is nul), maar wel door hoe goed de gematchte objecten die bij die boog horen overeenkomen. Het toevoegen van onvoorwaardelijke bogen is zeer belangrijk omdat er meer grote maximale cliques kunnen ontdekt worden die zoveel mogelijk objecten met elkaar matchen, ook al doet de onderlinge relatie van die extra gematchte objecten er niet toe. Men kan een analoge redenering maken voor het matchen van objecten, waarbij de binaire relaties tussen de gematchte objecten belangrijker zijn dan de matching van de objecten zelf. In dat geval worden er onvoorwaardelijke knopen aangemaakt in de association graph met een similarity score van nul. De similarity scores op de bogen worden wél op de gewone manier berekend.

We zullen nu enkele wiskundige uitdrukkingen bespreken waarmee we uiteindelijk een overall similarity score per maximale clique kunnen maken. Merk op dat dit niet hetzelfde is als de globale thresholding techniek! Het berekenen van een similarity score per maximale clique gebeurt om een rangschikking te bekomen van de oplossingen en dit gebeurt duidelijk ná het toepassen van alle lokale thresholds.

De *object similarity component* S_{obj} wordt berekend op basis van de similarity scores van alle gematchte object-paren. Deze paren zijn natuurlijk de knopen in de association graph. Zij M het aantal zulke object-paren en $i = 1 \dots M$. Zij o_i de similarity score van het i^{de} object-paar en $w_{o,i}$ een gewicht dat hoort bij het query-object van het i^{de} object-paar. Via de gewichten kan aan sommige objecten van de query meer belang worden toegekend. Het kan bijvoorbeeld voor de gebruiker meer belangrijk zijn dat een vierkant-object in de query gematcht wordt met een vierkant-object in de database dan dat een cirkel-object in de query gematcht wordt met een cirkel-object in de database. Het gewicht van het vierkant-object is dan hoger dan dat van het

cirkel-object.

S_{obj} wordt nu als volgt berekend:

$$S_{obj} = \frac{\sum_{i=1}^M w_{o,i} \cdot o_i}{\sum_{i=1}^M w_{o,i}} \quad (8.1)$$

Omdat geldt $\forall i : o_i \in [0,1]$, zal de teller van S_{obj} kleiner of gelijk zijn aan de noemer. Hierdoor geldt $S_{obj} \in [0,1]$.

De *relational similarity component* S_{rel} wordt berekend op basis van de similarity scores van de gematchte binaire relaties. Indien er M gematchte objecten zijn, dan zijn er $M(M-1)/2$ binaire relaties van de query die gematcht zijn. Laat $j = 1 \dots M(M-1)/2$. Zij r_j de similarity score van het j^{de} paar van geassocieerde binaire relaties en $w_{r,j}$ het gewicht dat hoort bij de j^{de} binaire relatie in de query. Dan wordt S_{rel} berekend als volgt:

$$S_{rel} = \frac{\sum_{j=1}^{M(M-1)/2} w_{r,j} \cdot r_j}{\sum_{j=1}^{M(M-1)/2} w_{r,j}} \quad (8.2)$$

Ook hier geldt dat $S_{rel} \in [0,1]$.

De gewichten o_i en r_j zijn een soort van globale instellingen van de gebruiker om het belang van bepaalde objecten respectievelijk relaties in de query aan te geven. Deze gewichten verschillen van de gewichten die door een applicatie kunnen gebruikt worden bij het aggregeren van een macro-score uit meerdere attribuut-scores waarover we tevoren kort gesproken hadden.

Objecten van de query en de database die niet gematcht worden zouden een soort van strafpunten moeten toevoegen waardoor de similarity score van de gehele matching lager wordt. We kunnen een afweging maken tussen het aantal objecten dat gematcht wordt en het aantal objecten dat niet gematcht wordt. Deze afweging duiden we aan met het begrip *scene completeness* en wordt gemodelleerd met de *scene completeness parameter* S_{comp} . Zoals tevoren, laat N het aantal objecten in de database aanduiden en n het aantal objecten in de query. De scene completeness parameter S_{comp} kan dan bijvoorbeeld als volgt berekend worden:

$$S_{comp} = \frac{M}{M + \alpha(n - M) + \beta(N - M)} \quad (8.3)$$

In bovenstaande uitdrukking is α een gewicht voor het aantal ongematchte query objecten en β een gewicht voor het aantal ongematchte database objecten. In uitdrukking (8.3) is elk query-

object even belangrijk en wordt gewoon de groep query-objecten als geheel gewogen met α . Maar hierboven hadden we ook vermeld dat er gewichten o_i kunnen gebruikt worden om meer belang toe te kennen aan bepaalde query-objecten. Indien een meer belangrijk query-object niet gematcht wordt, is dit "erger" dan wanneer een minder belangrijk query-object niet gematcht wordt. Dit geeft aanleiding tot een andere versie van de scene completeness parameter S_{comp} :

$$S_{comp} = \frac{\sum_{i=1}^M o_i}{\sum_{i=1}^M o_i + \alpha \sum_{i=M+1}^{n-M} o_i + \beta(N - M)} \quad (8.4)$$

De factor $(N - M)$ blijft behouden omdat de database-objecten geen gewichten hebben. Het is niet meteen duidelijk waarvoor gewichten voor database-objecten zouden gebruikt kunnen worden.

Men kan zien dat in beide uitdrukkingen $S_{comp} \in [0,1]$. Welk van de twee uitdrukkingen voor S_{comp} men wil gebruiken is afhankelijk van de toepassing.

Door te spelen met de waarden van parameters α en β in beide uitdrukkingen (8.3) en (8.4) zijn er verschillende belangrijke manieren mogelijk om een oplossing een score te geven op het vlak van scene completeness:

- 1) $\alpha = \beta = 1$: de scene completeness vindt ongematchte objecten van de query en ongematchte objecten van de database even belangrijk. Dit is waarschijnlijk het nuttigst wanneer de query en de database ongeveer evenveel objecten bevatten, denk bijvoorbeeld aan de populaire puzzelprenten "zoek de verschillen".
- 2) $\alpha = \beta = 0$: er zijn geen strafpunten als objecten niet gematcht worden, zowel voor de query als de database. In deze setting is men enkel geïnteresseerd in objecten die wél overeenkomen tussen de query en de database.
- 3) $\alpha = 1, \beta = 0$: enkel de niet-gematchte query-objecten worden in rekening gebracht. De aandacht wordt dus gevestigd op gematchte en niet-gematchte query objecten. Bij een GIS-toepassing kan het nuttig zijn om een zeer grote (continue) landkaart te queryen naar alle plaatsen die een meer bevatten met twee eilandjes in. In dit geval bevat de query drie objecten en de database mogelijk duizenden objecten.

Scenario's één en twee van hierboven produceren *symmetrische* resultaten omdat het dan niet uitmaakt welke ruimtelijke configuratie de query is en welke de database. Het derde geval is duidelijk *asymmetrisch*.

Nu gaan we alle componenten die we hierboven hebben geïntroduceerd samenvoegen tot een overall similarity-score voor een hele matching (clique), de zogenaamde *scene similarity* S_{scene} . Deze wordt als volgt berekend:

$$S_{scene} = S'_{scene} \left(w_{comp} (S_{comp} - 1) + 1 \right) \quad (8.5)$$

met

$$S'_{\text{scene}} = \frac{(w_{\text{obj}} \cdot S_{\text{obj}}) + (w_{\text{rel}} \cdot S_{\text{rel}})}{w_{\text{obj}} + w_{\text{rel}}} \quad (8.6)$$

Hierbij is w_{comp} het gewicht van de scene completeness parameter S_{comp} , w_{obj} het gewicht van de object similarity component S_{obj} en w_{rel} het gewicht van de relational similarity component S_{rel} . De uitdrukking voor S'_{scene} berekent een similarity score waar nog niet het effect van de scene completeness parameter in verwerkt is.

In (8.5) levert de uitdrukking $(S_{\text{comp}} - 1)$ waarden op in het interval $[-1, 0]$. Indien we aannemen dat $w_{\text{comp}} \in [0, 1]$, zal de uitdrukking $(w_{\text{comp}}(S_{\text{comp}} - 1) + 1)$ een waarde opleveren in het interval $[0, 1]$. Deze waarde kan de score S'_{scene} afzwakken: indien de scene completeness parameter laag is, zal de scene similarity score S_{scene} dus ook laag zijn.

De scene similarity score S_{scene} moet berekend worden voor elke maximale clique die we vinden in de association graph. Vervolgens worden al deze maximale cliques gesorteerd volgens stijgende S_{scene} -waarde. De best scorende clique komt overeen met een globaal optimum.

We zouden ook gewoon kunnen overwegen om een similarity score per maximale clique te berekenen door alle similarity scores op de knopen en bogen van zo'n clique op te tellen en het resultaat hiervan nemen als waarde voor S_{scene} . Ook in een dergelijke sommatie kan het gewicht van elk query-object en elke binaire query-relatie in rekening gebracht worden. In (Nedas & Egenhofer, Spatial-scene Similarity Queries, 2008) wordt aangegeven dat deze manier van werken een speciaal geval is van (8.6). Maar in cliques zijn er altijd veel meer bogen dan knopen, zeker in grotere cliques. In een clique met n knopen zijn er immers $O(n^2)$ bogen aanwezig. Hierdoor gaan de similarity scores van de relaties de similarity scores van de knopen overheersen! Daarom is de eerste manier van berekenen beter.

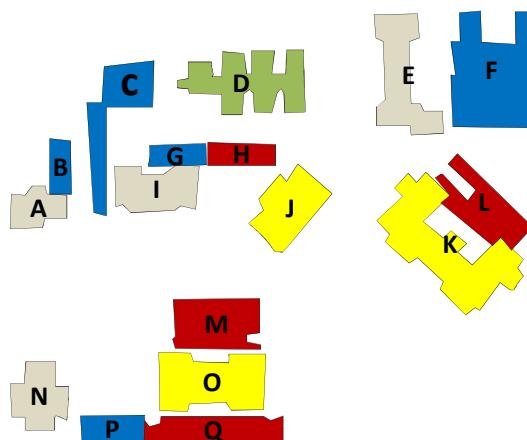
10.2.2. Voorbeeld query en oplossing

In (Nedas & Egenhofer, 2008) wordt een mooi en ook realistisch voorbeeld van een query-configuratie en database-configuratie gegeven. Daarmee wordt de werking van het matching-proces geïllustreerd. We hebben dat voorbeeld hier gereproduceerd en zelf wat aangevuld met extra opmerkingen. Dit voorbeeld bespreekt enkel het opstellen van de association graph en het zoeken naar maximale cliques. Het berekenen van een similarity score per maximale clique wordt niet behandeld.

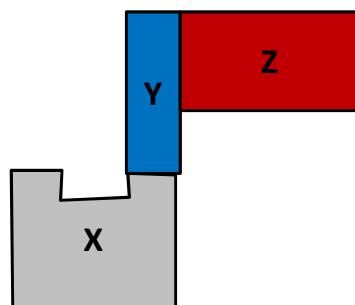
Beschouw de grote database-configuratie in Figuur 59 en de kleinere query-configuratie in Figuur 60. De database is een ruwe voorstelling van bepaalde gebieden op een campus. Elk object, aangeduid met een letter van A tot en met Q, is een gebied. De attributen per object zijn het type, gevisualiseerd met een kleur, en zijn veelhoekvorm. De objecten in de query, aangeduid met letters X-Y-Z, stellen ook gebieden voor die op een bepaald manier gepositioneerd zijn ten opzichte van elkaar. Ook deze objecten hebben een type-attribuut en een veelhoek-attribuut. Het is nu de bedoeling dat we alle deelverzamelingen van objecten uit de grotere database-configuratie kunnen vinden die een goede matching geven met de objecten uit de query. We eisen dat gematchte objecten van hetzelfde type moeten zijn. We mogen natuurlijk niet de binaire relaties vergeten die we óók moeten matchen. Specifiek voor dit voorbeeld wordt een binaire

relatie tussen twee objecten beschreven met een high-level topologisch predicaat uit paragraaf 8.1.1. Voor de eenvoud mag men zich inbeelden dat de lokale thresholds op alle knopen en bogen van de query-configuratie zó klein zijn dat het een exact-matching probleem wordt.

Laten we de association graph construeren. De eerste stap om dit te doen is voor elk object in de query alle objecten uit de database te verzamelen die voldoen aan zijn attributen. Voor dit voorbeeld betekent dit dat we voor een gebied-object uit de query alle gebied-objecten uit de database ophalen van hetzelfde type. Voor het vinden van gebieden van hetzelfde type kan men in principe een standaard SQL-query schrijven en ook standaard database indexatie-technieken gebruiken. Men kan natuurlijk in principe ook andere constraints toevoegen, zoals: match enkel gebieden waarvan de veelhoekvorm “gelijkt” op die van het query-object. Hiervoor kan een applicatie-specifiek algoritme ingeschakeld worden. Maar om het voorbeeld eenvoudig te houden zullen we enkel maar eisen dat het type van de gebieden overeenkomt.



Figuur 59: een database-configuratie van objecten die de gebieden voorstellen op een campus. Blauw: parkeerplaatsen van academisch personeel. Rood: residentiële parking. Grijs: academische gebouwen. De andere kleuren zijn niet belangrijk.



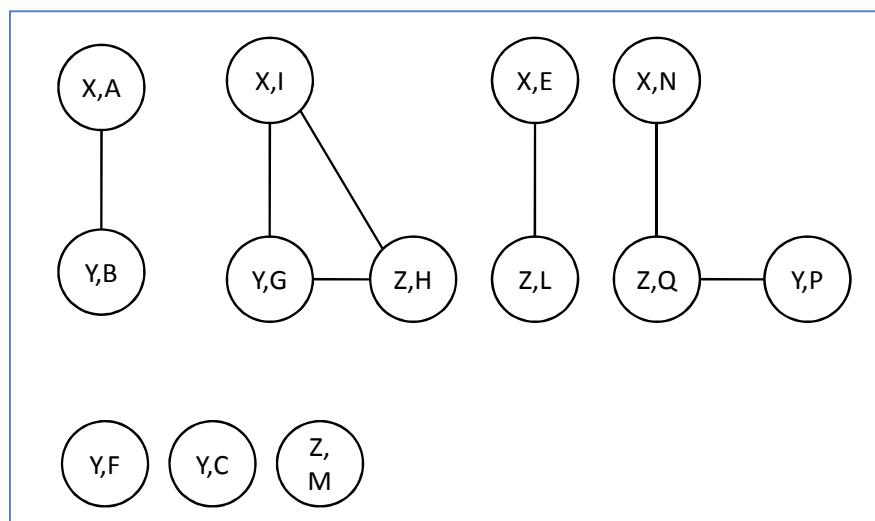
Figuur 60: een query-configuratie. Er is een academisch gebouw X dat raakt aan een personeelsparking, die zelf ook weer raakt aan een residentiële parking. Het academische gebouw raakt de residentiële parking niet.

Aangezien het type van gebied X uit de query “academisch gebouw” is, kan het gebied X gematcht worden met de database gebieden A, I, E en N omdat deze ook academische gebouwen voorstellen. Hierdoor worden de knopen (X,A), (X,I), (X,E) en (X,N) toegevoegd aan de association

graph. Op een vergelijkbare manier wordt gebied Y uit de query gematcht met de databasegebieden B, C, F, G, P. En gebied Z uit de query wordt gematcht met database-gebieden H, L, M en Q.

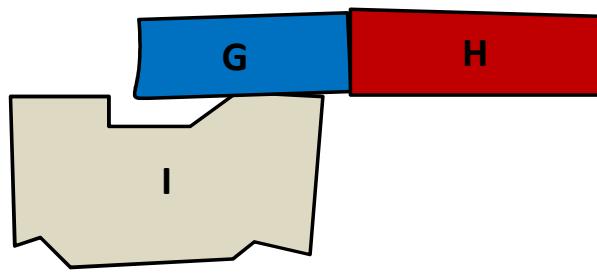
Nu worden alle mogelijke bogen beschouwd tussen knopen van de association graph en moet er gekeken worden of een boog is toegelaten of niet. Stel voor de eenvoud ook hier weer dat de lokale thresholds zo nauw zijn dat de relatie-attributen perfect moeten matchen. Bijvoorbeeld, de boog tussen knopen (Y,G) en (Z,H) is toegelaten omdat de applicatie-regels oordelen dat de relatie tussen gebieden Y en Z overeenkomt met de relatie tussen gebieden G en H. Maar de boog tussen (Y,G) en (X,A) kan niet toegelaten zijn omdat Y en X elkaar raken en G en A elkaar niet raken. Omdat we niet te veel bogen krijgen in de association graph, eisen we ook dat de afstanden tussen de gematchte objecten in de database ongeveer gelijk zijn aan de afstanden tussen de bijhorende objecten in de query. Dit is vooral handig als je wil eisen dat de mappings van objecten X en Z elkaar niet raken, maar ook niet al te ver uit elkaar mogen liggen!

De resulterende association graph wordt getoond in Figuur 61.



Figuur 61: association graph

We kunnen zien dat er een 3-clique is: $\{(X,I), (Y,G), (Z,H)\}$. Deze clique is de enige maximum clique. De verzameling objecten $\{I,G,H\}$ vormt een volledige match met de query-objecten en is gevisualiseerd in Figuur 62. Deelverzamelingen van deze 3-clique komen overeen met redundante oplossingen. De association graph bevat ook vier andere maximale cliques $\{(X,A), (Y,B)\}$, $\{(X,E), (Z,L)\}$, $\{(X,N), (Z,Q)\}$ en $\{(Z,Q), (Y,P)\}$ van grootte 2 en drie maximale cliques van grootte 1. Deze laatst genoemde negen cliques zijn allemaal onvolledige oplossingen. Men kan in principe een minimumgrootte van de cliques specifiëren zodat enkel maximale cliques beschouwd worden die minstens zoveel knopen bevatten als de minimumgrootte. Deze minimumgrootte kan bijvoorbeeld een bepaald percentage zijn van het aantal objecten in de query.



Figuur 62: de gematchte objecten uit de database-configuratie. Dit is een *volledige* oplossing voor de query in Figuur 60.

Indien men de lokale thresholds op de attributen van de objecten vergroot, is het mogelijk dat per object uit de query er meer objecten uit de database kunnen gematcht worden. Wanneer we vervolgens ook de thresholds op de relaties versoepelen, zullen er ook meer bogen verschijnen in de association graph. Indien er zeer sterk versoepeld wordt op beide soorten constraints, zal de association graph zeer veel knopen en bogen bevatten. Daardoor zullen er ook veel maximale cliques kunnen gevonden worden, misschien zelfs exponentieel veel in functie van het aantal knopen van de association graph. Eigenlijk is dit een realistisch gevolg van te sterke versoepelingen. Immers hoe minder eisen we stellen, hoe meer mogelijke oplossingen er mogelijk zijn! Meer constraints betekent in het algemeen dus een snellere uitvoering van de algoritmen maar wel minder mogelijke oplossingen. Elke applicatie moet dus goed overwegen hoe groot alle lokale thresholds mogen worden.

Afhankelijk van de applicatie is het mogelijk dat er verschillende lokale structuren bestaan in de association graaf. Een dergelijke lokale structuur is een verzameling knopen die geen bogen hebben naar knopen buiten deze verzameling. De maximale cliques zijn steeds een deelgraaf van slechts één lokale structuur. Bijgevolg is het mogelijk om te zoeken naar maximale cliques in parallel, waarbij elke thread een aparte lokale structuur krijgt toegewezen om in te zoeken.

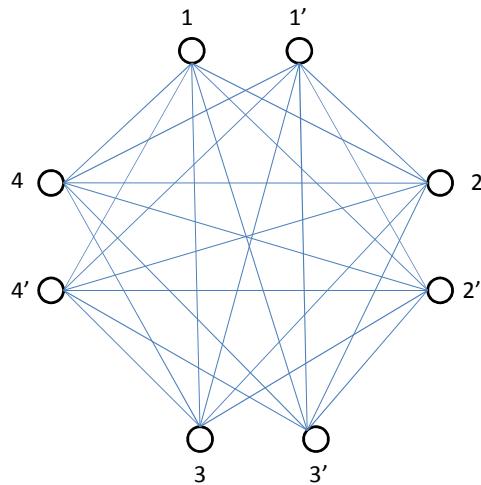
11. Maximal Clique finding

Dit hoofdstuk is gebaseerd op (Bron & Kerbosch, 1973), (Cazals & Karande, 2008) en (Koch, 2001). De notatie en pseudo-code zijn voornamelijk gebaseerd op (Cazals & Karande, 2008).

In dit hoofdstuk wordt een algoritme besproken dat vrij efficiënt alle maximale cliques in een graaf kan opsommen. Dit probleem wordt ook wel het *all-clique* probleem genoemd en het is NP-hard (Koch, 2001). Er wordt voor gezorgd dat cliques die in een andere clique bevatten niet apart worden gemeld door het algoritme. Het is de bedoeling dat we dit algoritme gebruiken voor het zoeken van maximale cliques in association graphs.

Het is in het algemeen mogelijk dat in een graaf exponentieel veel maximale cliques bevatten zodat er geen opsommingsalgoritme bestaat dat efficiënt is in functie van de grootte van de graaf. Beschouw bijvoorbeeld de graaf in Figuur 63. Deze graaf bevat vier paren van knopen: $(1, 1')$, $(2, 2')$, $(3, 3')$ en $(4, 4')$. De knopen binnen elk paar zijn niet verbonden door een boog. Elke maximale clique in deze graaf bevat dus nooit twee knopen van eenzelfde paar. Indien knoop 1 in een maximale clique zit bekomen we een andere maximale clique door knoop 1 te vervangen door knoop $1'$. Deze keuze kan per paar opnieuw herhaald worden zodat er 2^4 verschillende maximale cliques in deze graaf zitten. Indien er n zulke paren in een graaf zijn, zijn er dus 2^n maximale cliques, dus exponentieel veel in functie van de grootte van de graaf.

We proberen met een greedy heuristiek zo snel mogelijk alle maximale cliques te vinden, ook al zijn dit er in het slechtste geval exponentieel veel. Deze heuristiek zullen we het BK-algoritme noemen, naar de namen van de oorspronkelijke bedenkers: Bron en Kerbosch.



Figuur 63: voorbeeld-graaf

11.1. Basisalgoritme

Beschouw een ongerichte graaf $G = (V, E)$ met $|V| = n$. Beschouw een knoop $u \in V$, dan noteren we met $N[u]$ alle buren van deze knoop, dus $N[u] = \{v \in V \mid (u, v) \in E\}$. We laten niet toe dat een knoop een boog heeft naar zichzelf zodat $u \notin N[u]$.

Het BK-algoritme is recursief van aard en werkt met backtracking. We zullen eerst de eenvoudigste versie bespreken. De pseudo-code hiervan wordt getoond in listing 2. We zullen ruw schetsen wat er in het algoritme gebeurt. Het BK-algoritme houdt drie belangrijke verzamelingen

van knopen bij: $R \subseteq V$, $P \subseteq V$ en $X \subseteq V$. De verzameling R is de huidige clique die we aan het construeren zijn. Wanneer we afdalen in de backtracking-boom door het volgen van een tak, voegen we één knoop toe aan deze verzameling. De knopen die we *in principe* kunnen gebruiken om R uit te breiden naar een grotere clique bevinden zich in de verzamelingen P en X . P is de verzameling van kandidaten die we *effectief* mogen gebruiken om dit te doen, terwijl de knopen in X reeds werden gebruikt om R uit te breiden (waardoor al cliques zijn gevonden) en *niet* meer mogen gebruikt worden. Alle uitbreidingen van R waar minstens één knoop van X in zit zijn al beschouwd. Alle knopen in P en in X zijn per definitie buren van alle knopen in R . In het begin is R echter nog leeg en dan is vorige zin triviaal in orde. In verdere recursieve oproepen, wanneer R niet meer leeg is, geldt wegens *constructie* dat de knopen in P en X buren zijn van alle knopen in R . Hier komen we zodadelijk op terug bij het bespreken van de invarianten van het algoritme.

Indien de verzameling van kandidaten P in de huidige oproep van functie BK niet leeg is, is het de bedoeling dat we een kandidaat u_i in P selecteren en toevoegen aan R zodat we een grotere clique $R_{new} = R \cup \{u_i\}$ bekomen. Hierna mag de kandidaat u_i natuurlijk niet meer gebruikt

worden en op regel 10 van de pseudo-code wordt hij verwijderd uit P . Deze grotere clique R_{new} wordt meegegeven aan een volgende recursieve oproep op regel 14. De opmerkelijke "truc" van het BK-algoritme is om de verzameling van nieuwe kandidaten die wordt meegegeven aan deze volgende oproep te definiëren als $P_{new} = P \cap N[u_i]$. We houden enkel de kandidaten in P over die óók kinderen zijn van u_i . Daarnaast wordt ook een nieuwe verzameling van reeds gebruikte knopen gedefinieerd: $X_{new} = X \cap N[u_i]$. De definitie van P_{new} en X_{new} is om consistent te blijven met hun definitie, namelijk dat zij enkel knopen mogen bevatten die buren zijn van alle knopen in R_{new} . Nadat de recursieve oproep is afgelopen wordt de beschouwde kandidaat u_i in de verzameling X bijgevoegd. In een volgende iteratie van de for-loop wordt niet verdergewerkt met R_{new} maar terug met de gewone R waar u_i niet in zit. Alle cliques waar u_i in zit zijn op dat moment immers reeds beschouwd in diepere recursieve oproepen.

We mogen enkel een clique melden indien de verzameling P van kandidaten leeg is, want anders kan de clique R immers nog uitgebreid worden en zou de gemelde clique niet maximaal zijn.

Maar een lege verzameling P is geen voldoende voorwaarde om clique R te mogen melden, we hebben immers onze verzameling X nog. Hier komen we ook zodadelijk op terug bij het bespreken van de invarianten.

Om meer inzicht te krijgen in de werking van het algoritme, kan men in appendix 14.1 een gedetailleerde voorbeelduitvoering lezen van het algoritme in listing 2.

We beschouwen enkele invarianten van het algoritme (Koch, 2001):

- i. Alle knopen in R zijn paarsgewijs buren van elkaar, m.a.w. R is een clique.

In het begin is R nog leeg en dan is R een lege clique. Vervolgens wordt er een kandidaat $u_i \in P$ aan toegevoegd. Er worden daarna enkel knopen aan R toegevoegd die zelf buren zijn van alle knopen in R . Hierdoor is R dus steeds een clique, maar niet automatisch een maximale.

- ii. Elke knoop in V die alle knopen in R als buur heeft zit ofwel in P ofwel in X .

Deze uitspraak is makkelijk in te zien in de eerste oproep van het BK-algoritme omdat daar R nog leeg is. P is in het begin gelijk aan V en is X nog leeg. Na meerdere

iteraties van de for-loop in de eerste oproep zullen stilaan de knopen van V gepartitioneerd zijn in P en X omdat gebruikte kandidaten in X gestopt worden. Dit gebeurt trouwens in elke recursieve oproep, waardoor verzamelingen P en X altijd disjunct zijn.

Laten we veronderstellen dat de uitspraak voldaan is voor een bepaalde recursieve oproep.

Indien we noteren $V_R = \{v \in V \setminus R \mid R \subseteq N[v]\}$, dan geldt door deze aanname dat

$V_R = P \cup X$. Laten we R nu uitbreiden met een kandidaat u_i . We noteren met V_{R,u_i} de knopen die alle knopen in $R \cup \{u_i\}$ als buur hebben. Dan geldt:

$$\begin{aligned} V_{R,u_i} &\equiv V_R \cap N[u_i] \\ &= (P \cup X) \cap N[u_i] \\ &= (P \cap N[u_i]) \cup (X \cap N[u_i]) \\ &= P_{new} \cup X_{new} \end{aligned}$$

- iii. Beschouw een $u_i \in P$. Wanneer we u_i toevoegen aan X zijn alle cliques waar $R \cup \{u_i\}$ als deelverzameling in voorkomt reeds gemeld.

Wanneer we de uitbreiding $R \cup \{u_i\}$ beschouwen en een diepere recursieve oproep doen, geldt $P_{new} = \{u \in P \mid R \cup \{u_i\} \subseteq N[u]\}$. Alle buren van u_i die nog kunnen gebruikt worden om verdere uitbreidingen te doen worden dus meegestuurd naar de volgende oproep in de vorm van P_{new} . Door stilaan de kandidaten van P_{new} te verwerken worden uiteindelijk in alle diepere recursieve oproepen alle cliques minstens één keer gemeld waar $R \cup \{u_i\}$ als deelverzameling in zit. Bij terugkomst naar de huidige oproep wordt u_i overgeplaatst naar X .

- iv. Wanneer alle iteraties van de for-loop in een huidige oproep van het algoritme gedaan zijn, zijn alle cliques met R als deelverzameling precies één keer gemeld.

In het vorige puntje hebben we gezien dat $\forall u_i \in P$ geldt dat alle cliques met $R \cup \{u_i\}$ als deelverzameling minstens één keer gemeld worden. Omdat elke knoop $u_i \in P$ wordt overgeplaatst naar X na de recursie-stap, zal X alle knopen bevatten die reeds zijn beschouwd. Wanneer $P = \emptyset$ kan clique R niet meer uitgebreid worden. Er zijn dan twee mogelijkheden voor X :

- $X = \emptyset$. Beschouw de laatst beschouwde kandidaat u_i van het vorige niveau van recursie. We zullen dit vorige niveau van recursie A noemen en een kleine A schrijven in de superscript van zijn parameters. De X van het huidige niveau van recursie is de X_{new}^A van niveau A . Omdat $X_{new}^A = X^A \cap N[u_i] = \emptyset$, heeft kandidaat u_i dus geen knopen uit X^A als buren. Het is dus niet mogelijk dat er ooit een clique $R^A \cup \{u_i\}$ gemeld is door te vertrekken vanuit een knoop in X^A . Bijgevolg is de clique R van het huidige recursie-niveau een nieuwe maximale clique die gemeld mag worden.

- $X \neq \emptyset$. Dit gebeurt wanneer we alle $x \in X$ niet kunnen kwijtspelen op de manier die zojuist is beschreven. Elke diepere recursieve oproep waarbij $X \neq \emptyset$ zal nog niet voldoende “nieuwe” richtingen zijn uitgegaan om $X = \emptyset$ te krijgen. We weten dat alle knopen in X buren zijn van alle knopen in clique R . Een knoop $x \in X$ is per definitie al gebruikt om cliques mee te maken. Wanneer $P = \emptyset$, impliceert dit dat gehele clique R ook reeds beschouwd is bij het maken van alle cliques waar een $x \in X$ in zit en bijgevolg niet maximaal is en dus niet apart gemeld moet worden.

Listing 2: Maximal Clique Heuristic, version 1

```

1 input:  $V$  = set of vertices of graph  $G$ 
2 define  $\text{BK}(R, P, X)$  begin:
3   if  $P = \emptyset$  then
4     if  $X = \emptyset$  then
5       report maximal clique with nodes  $R$ 
6       return
7     else
8       Assume  $P = \{u_1, u_2, \dots, u_k\}$ 
9       for  $i \leftarrow 1$  to  $k$  do
10       $P = P - \{u_i\}$ 
11       $R_{\text{new}} = R \cup \{u_i\}$ 
12       $P_{\text{new}} = P \cap N[u_i]$ 
13       $X_{\text{new}} = X \cap N[u_i]$ 
14       $\text{BK}(R_{\text{new}}, P_{\text{new}}, X_{\text{new}})$ 
15       $X = X \cup \{u_i\}$ 
16    end
17 initial call:  $\text{BK}(\emptyset, V, \emptyset)$ 

```

11.2. Uitbreidingen

We zullen nu enkele trucjes beschouwen waardoor we ons basisalgoritme uit vorige paragraaf sneller kan gemaakt worden.

Beschouw R, P en X van de huidige oproep van het BK-algoritme. Beschouw een bepaalde kandidaat $u_t \in P$. Indien we deze toevoegen aan R en verdere recursieve oproepen afwerken, zullen alle cliques gemeld worden waarin u_t zit en mogelijk daarbij ook zijn buren in $N[u_t] \cap P$.

Stel dat u_t vóór elk van zijn buren $u_i \in N[u_t] \cap P$ wordt toegevoegd aan R . In alle diepere

recursieve oproepen die verder uitbreiden aan clique $R \cup \{u_t\}$ zullen ook alle buren $u_i \in N[u_t] \cap P$ worden beschouwd om de clique uit te breiden. Heeft het zin om voor elke u_i nog eens apart een nieuwe recursie-tak te starten om clique $R \cup \{u_i\}$ uit te breiden? Indien een buur u_i geen andere nieuwe buren heeft in P t.o.v. u_t , dus $P \cap (N[u_i] \setminus N[u_t]) = \emptyset$, zal de oproep om $R \cup \{u_i\}$ uit te breiden nooit eindigen met het melden van een nieuwe maximale clique omdat dan steeds zal gelden: $u_i \in X$. Hierdoor zal de voorwaarde voor het melden van een maximale clique nooit voldaan zijn omdat $X \neq \emptyset$. We mogen dus alle buren u_i van u_t in P snoeien waarvoor geldt $P \cap (N[u_i] \setminus N[u_t]) = \emptyset$.

Maar indien $P \cap (N[u_i] \setminus N[u_t]) \neq \emptyset$, dan bestaat er een $u_k \in P : u_k \in N[u_i] \wedge u_k \notin N[u_t]$ waar we in een diepere recursie-tak, vertrekende vanuit de uitbreiding $R \cup \{u_i\}$, nieuwe maximale cliques mee zouden kunnen maken die verschillen van alle maximale cliques waar u_t inzit. Omdat $u_t \notin N[u_k]$ zal u_t wél uit de verzameling X worden gehaald, namelijk bij het uitbreiden van de clique met de kandidaat u_k . Wat gebeurt er indien we een buur u_i snoeien terwijl deze zelf een buur $u_k \in P \cap (N[u_i] \setminus N[u_t])$ heeft? Missen we dan een kans op het maken van bepaalde nieuwe cliques? Neen, omdat in de huidige oproep in een bepaalde iteratie van de for-loop toch u_k nog beschouwd wordt, en omdat hij geen buur is van u_t laten we hem met rust. We breiden vervolgens R uit met u_k en doen een nieuwe recursieve oproep. Deze nieuwe recursie-tak is dan de tak die we gesnoeid hadden door geen oproep te doen voor buur u_i . We beschouwen de uitbreiding van de clique met kandidaat u_k dus gewoon op een hoger niveau.

We verliezen dus geen kansen door R nooit uit te breiden met zijn buren $u_i \in N[u_t] \cap P$. We hebben hierboven de redenering opgebouwd door te doen alsof u_t vóór elk van zijn buren $u_i \in N[u_t] \cap P$ als uitbreiding werd beschouwd van R . Maar eigenlijk is dit helemaal niet nodig, volgorde speelt geen rol. Indien we u_t helemaal vanachter in P zouden selecteren en zijn buren u_i zouden negeren die vóór u_t in de for-loop beschouwd worden, dan is het netto resultaat hetzelfde als wanneer we éérst u_t zouden beschouwd hebben. Want voor een genegeerde buur u_i maakt het toch niet uit of u_t al dan niet in X aanwezig was.

Knoop u_t wordt de *pivot* genoemd. In (Koch, 2001) wordt voorgesteld om de pivot te kiezen met het meeste aantal buren $u_i \in N[u_t] \cap P$. Hierdoor zal de for-loop dus het meeste aantal kandidaten kunnen snoeien. Dit wordt de *greedy* manier genoemd en deze blijkt in de praktijk het meeste kans op snelheidswinst te kunnen bieden. Maar in sommige gevallen kan de efficiëntie van deze keuze wel erger zijn dan een willekeurige selectie (Cazals & Karande, 2008). Maar laten we voor de eenvoud de greedy aanpak volgen omdat we toch niet eenvoudig kunnen voorspellen hoe de precieze vorm van de association graph zal zijn.

Beschouw R , P en X van de huidige oproep van het BK-algoritme. Stel dat we nu pivots mogen kiezen uit $P \cup X$. Dan is het mogelijk dat de pivot een knoop $x \in X$ is. Indien we nu in de for-

loop de buren van x negeren, dus alle $u_i \in N[x] \cap P$, verliezen we dan de kans op het maken van sommige cliques? We mogen sowieso alle knopen u_i negeren waarbij

$P \cap (N[u_i] \setminus N[x]) = \emptyset$ omdat bij het maken van alle cliques waarin x zit toch deze u_i werden beschouwd om op te nemen in deze cliques. Indien er een u_i bestaat waarbij

$P \cap (N[u_i] \setminus N[x]) \neq \emptyset$, dan volgen we opnieuw dezelfde redenering als hierboven. Knoop $u_k \in P \cap (N[u_i] \setminus N[x])$ wordt niet genegeerd omdat hij geen buur is van de pivot. De clique zonder x met als deelverzameling $R \cup \{u_i, u_k\}$ wordt dan misschien ergens in een recursieve tak gemaakt die begint met de uitbreiding van R met u_k . We besluiten dat de pivot dus evengoed afkomstig mag zijn uit X .

Fixeer binnen de huidige oproep van het BK-algoritme één knoop $x \in X$. Per definitie van verzameling X hebben we reeds alle maximale cliques gemeld waar x in zit. Daarnaast is per definitie x ook buur van alle knopen in R . Stel nu dat x buur is van alle knopen in P , dus $P \subseteq N[x]$. Alle cliques die we vanuit de huidige recursie-oproep kunnen maken bevatten dus allemaal knopen waar x buur van is. Maar omdat $x \in X$ we hebben reeds alle cliques gemeld waar deze knopen in zitten. Bijgevolg kunnen we nu de huidige oproep afbreken omdat we x nooit uit X zullen kunnen verwijderen in diepere recursieve oproepen en dus nooit R zullen mogen melden in het basisgeval $P = \emptyset$. Men zou het geval $\exists x \in X : P \subseteq N[x]$ apart kunnen detecteren aan het begin van elke recursieve oproep of na elke verandering van de verzameling X in de for-loop. Maar dat is eigenlijk niet nodig indien we de pivot mogen kiezen uit $P \cup X$. Beschouw $x \in X$ zodat $P \subseteq N[x]$. In dat geval geldt $\forall u_i \in P : |P \cap N[u_i]| < |P \cap N[x]|$. Deze strikte ongelijkheid geldt omdat een knoop nooit zichzelf als buur kan hebben. Indien we de greedy aanpak voor pivot-selectie volgen zal x als pivot geselecteerd worden en zullen bijgevolg alle kandidaten in P genegeerd worden omdat ze allemaal buren zijn van x . Op een elegante manier worden via deze pivot-selectie dus verdere oproepen afgebroken. Indien we niet de greedy aanpak volgen kunnen we best het geval $\exists x \in X : P \subseteq N[x]$ wel apart detecteren.

Maar de pivot wordt slechts éénmaal geselecteerd per oproep en niet dynamisch na elke verandering aan X . Het is mogelijk dat we pas na een aantal iteraties de situatie krijgen $\exists x \in X : P \subseteq N[x]$ waarbij P al geslonken is en X gegroeid doordat kandidaten beschouwd zijn. Er zal voor elk van deze kandidaten in P nog maximaal één recursieve oproep gebeuren als zij niet directe buren zijn van de pivot. Want omdat geldt dat $\exists x \in X : P \subseteq N[x]$, geldt ook dat $\exists x \in X_{new} : P_{new} \subseteq N[x]$. In de recursie-stappen met deze P_{new} en X_{new} zal meteen aan het begin van de for-loop de pivot uit X_{new} geselecteerd worden zodat bovengenoemde snoei-proces toch begint, slechts met één recursie-niveau vertraging.

Indien we de pivot-selectie en het negeren van zijn buren nu toevoegen aan het eerste BK-algoritme bekomen we versie 2, getoond in listing 3. Deze tweede versie genereert de maximale cliques in een onvoorspelbare volgorde om het aantal takken van de backtracking-boom te verminderen. Natuurlijk worden wel dezelfde cliques gemeld als in het eerste algoritme. De auteurs in (Bron & Kerbosch, 1973) zeggen dat deze versie de neiging heeft om de grootste cliques eerst te melden en ook cliques die een grote gemeenschappelijke doorsnede hebben in

volgorde. Het kernpunt van deze nieuwe versie is dus het verminderen van het aantal keer dat er nieuwe recursieve oproepen moeten gestart worden vanuit eenzelfde for-loop door het snoeien van kandidaten via de pivot.

Listing 3: Maximal Clique Heuristic, version 2, with pivot selection

```

1   input:  $V$  = set of vertices of graph  $G$ 
2   define  $\text{BK}(R, P, X)$  begin:
3       if  $P = \emptyset$  then
4           if  $X = \emptyset$  then
5               report maximal clique with nodes  $R$ 
6               return
7           else
8               select pivot  $u_i$  // zie paragraaf 11.2
9               Assume  $P = \{u_1, u_2, \dots, u_k\}$ 
10              for  $i \leftarrow 1$  to  $k$  do
11                  if  $u_i \notin N[u_i]$  then
12                       $P = P - \{u_i\}$ 
13                       $R_{new} = R \cup \{u_i\}$ 
14                       $P_{new} = P \cap N[u_i]$ 
15                       $X_{new} = X \cap N[u_i]$ 
16                       $\text{BK}(R_{new}, P_{new}, X_{new})$ 
17                       $X = X \cup \{u_i\}$ 
18      end
19  initial call:  $\text{BK}(\emptyset, V, \emptyset)$ 

```

11.3. Bespreking implementatie

In deze paragraaf bespreken we kort een aantal resultaten die we hebben bekomen met een eigen Java-implementatie van listing 3. Hieruit moet blijken dat het een zeer efficiënt algoritme is. We geven de details van het gebruikte testplatform:

- Intel® Core™2 Duo CPU T7100, 1.80GHz
- Windows XP
- JDK 1.6

In onderstaande tabel worden de resultaten samengevat. Er zijn vier experimenten uitgevoerd, telkens voor een andere maximale hoeveelheid knopen in de graaf. Voor elke graafgrootte werden 10 instanties gegenereerd. In elk van deze 10 instanties werd gezocht naar maximale cliques. De minimale, maximale en gemiddelde uitvoeringstijden werden opgemeten over deze 10 grafen heen.

Het aantal bogen van alle 40 grafen was steeds minstens de helft van het totaal aantal mogelijke bogen, waardoor deze grafen dus niet zo sparse zijn en mogelijk veel maximale cliques bevatten.

Deze grafen werden op een random manier gegenereerd met een naïef algoritme dat zelf werd bedacht: er worden met een random number generator paren identifiers van knopen gegenereerd. Tussen deze knopen wordt dan een boog getrokken indien deze er nog niet was. Deze manier is zeer traag op het vermelde testplatform, namelijk meer dan 2 minuten om één graaf te genereren met 200 knopen erin. Maar het belangrijkste zijn natuurlijk de tijdmetingen voor het Bron-Kerbosch algoritme!

Tijdens de uitgevoerde experimenten werd het algoritme van listing 3 reeds gestopt na het vinden van vier cliques, zodat er niet gezocht wordt naar een exponentieel aantal maximale cliques.

Tijdens een praktisch gebruik van een maximal clique finding algoritme in de context van hoofdstuk 10 zal men vermoedelijk meestal genoeg hebben aan het teruggeven van een constant aantal goede oplossingen. In (Bron & Kerbosch, 1973) wordt ook opgemerkt dat de grootste maximale cliques de neiging hebben om het eerst gevonden te worden door het algoritme.

Gemiddeld aantal knopen in graaf	Aantal instanties van de graafgrootte	Minimum Zoektijd (milliseconden)	Maximum Zoektijd (milliseconden)	Gemiddelde Zoektijd (milliseconden)
9	10	0.0483	0.7386	0.1350
30	10	0.4006	6.3720	1.4774
100	10	11.0657	312.5364	48.1114
200	10	82.0258	8286.8660	1115.2366

We zien in de tabel duidelijk dat voor grotere grafen de zoektijd toeneemt. Voor een graaf met gemiddeld 200 knopen erin kan de zoektijd soms al oplopen tot 8 seconden, maar eigenlijk is dit nog vrij snel aangezien een brute-force algoritme veel slechter scoort. Een brute-force algoritme dat alle mogelijke deelverzamelingen van een graaf uitprobeert en test of deze een clique zijn produceert naar verwachting zeer slechte resultaten. Bijvoorbeeld, op een graaf van ongeveer 100 knopen is de zoek-tijd om één clique te vinden opgelopen tot meer dan 15 minuten. Het algoritme werd vroegtijdig beëindigd. Bovenstaande resultaten voor het Bron-Kerbosch algoritme zijn dus zeer goed in vergelijking met brute-force technieken.

12. Perspectief op deel 2

In (Nedas, 2006) wordt aangegeven dat het moeilijk lijkt om similarity tussen twee complexe (ruimtelijke) configuraties of structuren te beoordelen met een computer. Maar het is mogelijk om het complexe probleem te ontbinden in eenvoudig begrijpbare operaties op kleinere delen. Het algoritme om similarity vast te stellen werkt dan een beetje in een bottom-up manier waarbij similarity-informatie wordt verzameld op de laagste niveaus en geleidelijk door aggregation steeds meer hogere niveau vaststellingen kunnen gedaan worden. Dit is inderdaad wat we in het theoretische raamwerk van hoofdstuk 10 hebben bestudeerd, in de context van GIS. Daar hadden we ruimtelijke objecten in een query-configuratie en ruimtelijke objecten in een database-configuratie. De eerste eenvoudig begrijpbare taak was het vinden van individuele overeenkomstige objecten tussen de twee configuraties. De tweede eenvoudig begrijpbare taak was controleren of bepaalde verbanden tussen de objecten in de query ook gelden tussen de objecten uit de database. Na deze twee stappen bekwamen we de association graph. De derde eenvoudig begrijpbare taak was het zoeken naar cliques in deze graaf.

In de literatuur zijn momenteel veel applicatie- en domein-specifieke technieken ontwikkeld om similarity vast te stellen. We hebben bijvoorbeeld het specifieke wavelet algoritme in hoofdstuk 7 besproken voor image querying en de aanpak van complexe netwerken om shapes te herkennen in hoofdstuk 9. Indien een toepassing enkel gebruik maakt van deze specifieke algoritmen lijkt dit slechts een beperkte bewegingsvrijheid te geven omdat men minder eenvoudig nieuwe technieken kan innpassen. De opmerking van (Nedas, 2006) hierboven is zéér hoopgevend omdat dat we het ogenschijnlijk zeer complexe probleem van similarity searching en pattern matching toch op een elegante manier kunnen aanpakken met meer theoretisch onderbouwde technieken. Daarom hebben we besloten deel 2 van deze thesis te wijden aan het verder uitwerken van hoofdstuk 10, vooral omdat we een abstractie kunnen maken van applicatie-specifieke regels. Hieronder staan een aantal (mogelijke) plannen opgeschreven.

12.1. Beschrijving eigen applicatie

Concreet zijn we geïnteresseerd in het (semi-)automatisch ontdekken van visuele overeenkomsten tussen twee afbeeldingen door gebruik te maken van association graphs en maximal clique finding. Het ontdekken van visuele overeenkomsten op deze manier is een zelf verzonnen toepassing van de algemene theoretische aanpak met association graphs uit hoofdstuk 10. Maar blijkbaar bestaat hierover reeds literatuur: (Horaud & Skordas, 1989). Dit bevestigt alleen maar dat de techniek werkt.

In (Horaud & Skordas, 1989) wordt het probleem van *stereo correspondence* aangepakt. Dit probleem is het zoeken van overeenkomsten tussen twee afbeeldingen die dezelfde scène voorstellen, maar gezien vanuit een linker- en rechter-camera. Deze opstelling van camera's moet het menselijk dieptezicht nabootsen. Door gebruik te maken van eigenschappen van de camera's zelf en door het lichte verschil tussen beide afbeeldingen kan men (benaderend) afleiden hoe ver objecten zich van de camera-opstelling bevinden, waardoor een soort dieptezicht ontstaat. Men zet hiertoe eerste beide afbeeldingen om naar een edge-map met algoritmen uit de beeldverwerking. Een edge-map is een verzameling polylines (edges) in het vlak die aangeven waar zich belangrijke kleurveranderingen of intensiteitsveranderingen voordoen in de afbeelding, een beetje zoals we gezien hebben bij de 2D wavelet-coëfficiënten in hoofdstuk 7. Deze edges worden in rechte lijnstukken geknipt. Deze lijnstukken komen overeen met ruimtelijke objecten, zodat elke afbeelding kan beschouwd worden als een configuratie van lijnstukken met *binaire* relaties tussen deze lijnstukken. Het is vervolgens de bedoeling dat er een association graph wordt opgesteld van beide configuraties om te zoeken naar verbanden. Dit is een mooie toepassing van de meer algemene theorie van paragraaf 10.2. In paragraaf 10.2 hebben we immers aangegeven dat er vele soorten ruimtelijke objecten kunnen beschouwd worden en vele soorten binaire relaties tussen deze objecten, allemaal beslissingen die afhankelijk zijn van de concrete toepassing.

We zijn in deze thesis niet specifiek geïnteresseerd in het stereo correspondence probleem, maar wel in het zoeken naar overeenkomsten tussen twee volledig willekeurige afbeeldingen. Er zijn veel mogelijke toepassingen denkbaar voor een dergelijk algemeen image-matching algoritme:

- 1) Zoeken naar gelijkaardige afbeeldingen, zoals tegenwoordig mogelijk is op verscheidene websites, waaronder Google Similar: <http://similar-images.googlelabs.com/>.
- 2) Herkenning van voorwerpen in foto's: het herkennen van voorwerpen in foto's zou kunnen gebeuren door template-patronen van de te herkennen voorwerpen als queries aan te bieden aan de foto. De foto fungeert dan als grotere database waarin een kleinere query-configuratie moet gezocht worden. Het basisidee hiervan is al besproken in het voorbeeld van paragraaf 10.2.2.
Een concrete voorbeeldtoepassing zou kunnen zijn: het herkennen van mensengezichten in foto's of letters/symbolen in oude manuscripten. Een ander voorbeeld is het zoeken naar vliegtuigen op satellietfoto's van luchthavens.

We zijn geïnteresseerd in beide toepassingen, maar vooral in de eerste. Hoofdstuk 7 gaat al over dit onderwerp, maar wavelets hebben hun beperkingen, zoals aangegeven in het begin van hoofdstuk 10. Over image retrieval/querying in het algemeen is een survey-paper geschreven die we kunnen doornemen: (Ritendra Datta, 2008). In deze survey worden allerlei image retrieval technieken van de afgelopen jaren samengevat.

We hopen dat door het maken van een graaf-abstractie van een afbeelding en de techniek van association graphs het mogelijk zal zijn om twee afbeeldingen als gelijkaardig te beschouwen indien de ene een elastische deformatie is van de andere.

12.2. Overzicht van mogelijke technieken

Allerlei specifieke technieken kunnen gebruikt worden om de association graph, vermeld in hoofdstuk 10, op te stellen. Om efficiënt vast te stellen welke objecten met elkaar gematcht kunnen worden zal het ook interessant zijn om concrete indexatie-technieken voor attribuut-waarden te bestuderen. Hieronder vallen geometrische vormen, kleuren, Spatial access methods zoals quadtrees zijn ook niet ondenkbaar om efficiënt punten en objecten in een bepaald gebied van het vlak op te zoeken.

We zullen nu even aangeven welke mogelijkheden er bestaan om een afbeelding om te zetten naar een ruimtelijke configuratie van objecten en relaties tussen deze objecten.

We kunnen er misschien voor kiezen om een afbeelding om te zetten naar een edge map. Hiervoor zal vermoedelijk een image processing library ingeschakeld worden of we veronderstellen in een initiële faze van de ontwikkeling dat de edge maps al beschikbaar zijn als verzamelingen van polylines. Het zal de bedoeling zijn om in deze verzamelingen polylines concrete features te ontdekken die we kunnen gebruiken als ruimtelijke objecten. Dit moet gebeuren voor beide afbeeldingen waartussen we similarities willen ontdekken. Concreet is het in feite mogelijk om de polylines zelf te beschouwen als ruimtelijke objecten. Het is mogelijk dat met specifieke technieken uit de literatuur belangrijke features van deze polylines kunnen afgeleid worden. Het is dan interessant om te overwegen hoe binaire relaties tussen deze features of tussen de polylines kunnen gedefinieerd worden. Het is mogelijk dat hiervoor computational geometry kan gebruikt worden, zoals bijvoorbeeld Delaunay Triangulations. (Constrained) Delaunay Triangulations worden gebruikt om *neighborhoods* van spatial objects te berekenen en te kijken welke objecten dicht bij elkaar liggen (Blaser, 2000).

Daarnaast is het ook mogelijk om de kleurinformatie van de oorspronkelijke afbeelding die zich bevindt *tussen* de polylines te gebruiken in de similarity assessment. Gekleurde vlakken kunnen dan ook ruimtelijke objecten worden die een bepaalde veelhoekvorm hebben en eventueel zelfs een texture. Er moet dan onderzocht worden hoe veelhoeken efficiënt kunnen gematcht worden en hoe hun kleur en textuur hierbij een rol kan spelen. Daarnaast is het ook belangrijk dat polyline-

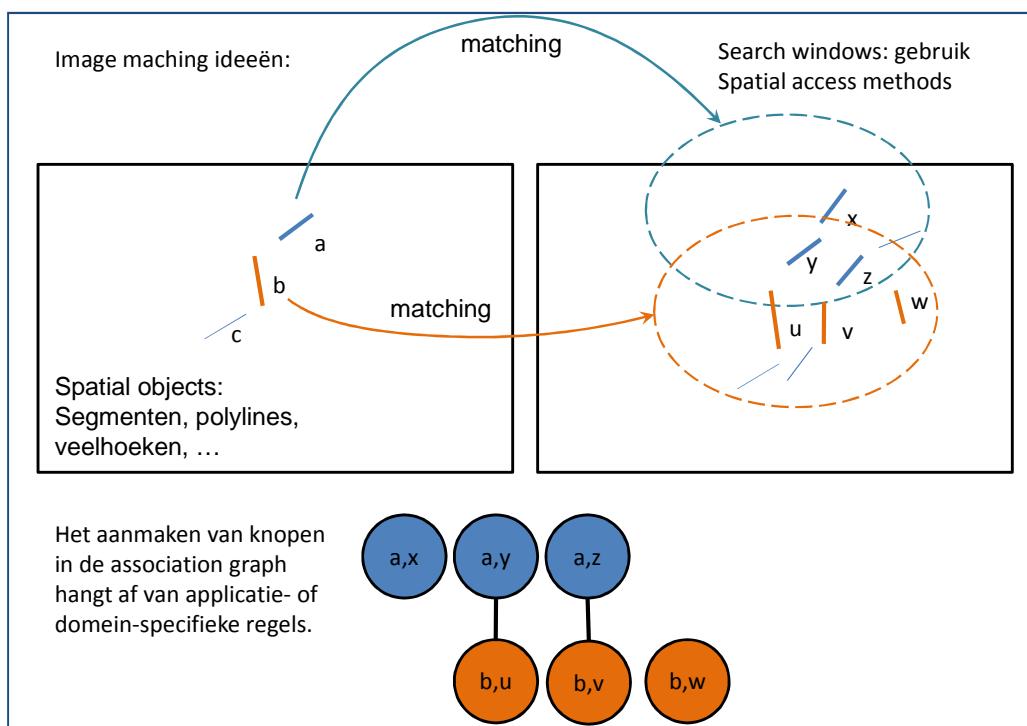
objecten bepaalde binaire relaties hebben met deze gekleurde vlakken.

In plaats van edge maps is het misschien ook mogelijk om met beeldverwerkingstechnieken lokale feature points te extraheren. Deze feature points kunnen dan fungeren als ruimtelijke objecten.

Tussen feature points kunnen we dan binaire relaties beschouwen. Hiertoe moet eerst nog literatuur voor bestudeerd worden.

In (Egenhofer, 1997) wordt aangegeven hoe high-level topologische relaties kunnen uitgebreid worden met kwantitatieve informatie zoals afstanden en oppervlakten. Afhankelijk van welke informatie we hiervan gebruiken kunnen we similarity queries uitvoeren op verschillende niveaus van detail. Concreet wordt in het vernoemde werk eerst gekeken of de high-level topologische relatie tussen twee veelhoeken is voldaan. Indien dit voor een paar veelhoeken in de database niet het geval is, kan die oplossing reeds gesnoeid worden. Door steeds meer gedetailleerde aspecten van similarity toe te passen kan er geleidelijk gesnoeid worden in de oplossingsruimte. Het kan interessant zijn om in de eigen applicatie te kijken naar de effecten van verschillende detaileigenschappen van een binaire relatie op het terugvinden van oplossingen.

In onderstaande afbeelding wordt heel conceptueel aangegeven hoe het matchen van images zou kunnen verlopen. Deze afbeelding is enkel gemaakt om bovenstaande ideeën lichtjes te illustreren en vormt geen beperking op de mogelijkheden.



Het kan eventueel nog interessant zijn ook heuristieken te leren kennen voor het maximal clique finding probleem, zoals bijvoorbeeld (Sutera, Abu-Khzam, Zhang, Symons, Samatova, & Langston, 2005). Maar we kunnen starten met het eenvoudige Bron-Kerbosch algoritme uit hoofdstuk 11 omdat deze veel gebruikt wordt in de praktijk (Cazals & Karande, 2008).

Er zijn dus duidelijk twee onderdelen met een verschillende invalshoek. Het opstellen van de association graph zal vereisen dat er meer domein-specifieke technieken worden geïmplementeerd. Het zoeken naar maximale cliques is een zeer algemeen theoretisch probleem en kan volledig onafhankelijk geïmplementeerd worden. Dit is meteen een mogelijke sterkte van de geplande implementatie omdat beide onderdelen onafhankelijk kunnen onderzocht en geoptimaliseerd kunnen worden. De implementatie, zoals hierboven geschatst, zal een mooie samensmelting zijn van theoretische informatica met domein specifieke kennis. Helaas is het zoeken naar cliques een NP-compleet probleem, maar het heeft dus duidelijk wel praktische relevantie.

Al deze genoemde technieken en ideeën zijn uiteindelijk belangrijk voor een applicatie, maar voor deze thesis is het misschien interessant om in het begin een eenvoudig algoritme te implementeren of te gebruiken om een afbeelding om te zetten naar ruimtelijke objecten en dan meteen verder te gaan met databasegeoriënteerde topics die reeds mogen veronderstellen dat *op de een of andere manier* ruimtelijke configuraties zijn bekomen. Maar we zullen ons eerst sowieso concentreren op het uitwerken van een eenvoudig image matching algoritme en hierop verderbouwen in de context van database technieken voor image retrieval. Andere theoretische onderwerpen die immers aan dit onderwerp vasthangen zijn het clusteren van graphs in een database (Conte, Foggia, Sansone, & Vento, 2007). Dit kan waarschijnlijk ook gebruikt worden bij similarity searching: maak eerst clusters van gelijkaardige grafen. De grafen in een cluster stellen dan allemaal gelijkaardige afbeeldingen voor. Gelijkheid op grafen kan dan in een preprocessing stap afgeleid worden met maximal clique finding op association graphs en wanneer deze cliques een score toegekend krijgen op een manier zoals bestudeerd in paragraaf 10.2.1.1, kunnen grafen waartussen veel overeenkomsten zijn in eenzelfde cluster worden ondergebracht.

Tenslotte kan het in een bepaald algoritme, zoals gezien in hoofdstuk 7, gebeuren dat er vrij veel parameters aanwezig zijn om het resultaat te *tunen*. Het is soms zeer lastig om parameters met de hand in te stellen. Een bepaalde techniek die in machine learning gebruikt wordt is het geven van voorbeeld-inputs met bijhorende outputs. Deze paren vormen constraints op de parameters en via least-squares methoden kunnen de parameters dan *gefit* worden om zo goed mogelijk te voldoen aan deze constraints. Om meer over dit onderwerp te weten te komen kunnen we het boek (Mitchell, 1997) bekijken.

12.3. Eventuele theoretische uitbreidingen

Men geeft in de literatuur aan (Egenhofer, 1997) dat met binaire relaties alles omtrent onderlinge posities van objecten kan uitgedrukt worden. Na zelf wat nagedacht te hebben, leek het echter mogelijk dat sommige relaties tussen meerdere groepen objecten in een ruimtelijke configuratie enkel konden uitgedrukt worden door *n*-aire relaties te beschouwen, in de vorm van graafstructuren in het vlak. Maar misschien kan dit toch uitgedrukt worden met applicatie-specifieke oplossingen tijdens het construeren van de association graph. We zullen voorlopig proberen enkel binaire relaties te gebruiken en indien er grote praktische problemen ontstaan bij de bovengenoemde implementatie kan eventueel onderzocht worden of *n*-aire relaties hier enige oplossing voor zouden kunnen bieden, maar dit is zeker niet triviaal.

Een andere mogelijkheid is onderzoeken of de association graph volledig moet geïnstantiëerd worden. Misschien kunnen datamining technieken helpen beslissen of bepaalde delen van deze graaf wel of niet waarschijnlijk maximale cliques kunnen bevatten.

Andere vragen die we kunnen stellen zijn:

- 1) Moeten we per se cliques uit de association graph halen? Wat gebeurt er als we bijvoorbeeld bomen of niet-volleig geconnecteerde deelgrafen uit de association graph halen? Wanneer we cliques beschouwen is de mapping van query-objecten naar database-objecten één-op-één. Indien we een willekeurige boom zouden selecteren uit de association graph is het mogelijk dat één query-object wordt gemapt op meerdere database-objecten (*split*) of dat meerdere query-objecten worden gemapt op één database-object (*merge*). Bij unconstrained Dynamic Time Warping bijvoorbeeld zijn gelijkaardige multi-multi mappings mogelijk tussen de samples van de sequenties die men probeert te aligneren (Kellens, 2006).
- 2) Misschien is het ook mogelijk om maximale cliques te ontdekken door ook rekening te houden met de gewichten op de edges? We zoeken dus naar goed scorende oplossingen *terwijl* we de cliques aan het zoeken zijn. Dit past wel bij het vorige puntje waar we

bomen zouden kunnen extraheren uit de association graph die een similarity score maximaliseren.

- 3) Kan de topologie van de association graph willekeurig zijn, m.a.w. is elke graaf een association graph? Indien dit niet zo blijkt te zijn kan eventueel gezocht worden naar een efficiënter clique-finding algoritme dat bepaalde eigenschappen van de grafen kan uitbuiten. Vermoedelijk is de topologie van een association graph echter willekeurig en zelfs afhankelijk van de applicatie.

Deze vragen kunnen eventueel verder uitgewerkt worden indien zij zich stellen tijdens verdere uitwerking van de eigen implementatie.

12.4. Planning

In deel 2 van deze thesis zitten duidelijk nog een aantal literatuurstudies. We denken voorlopig de topics van deel 2 als volgt in te plannen:

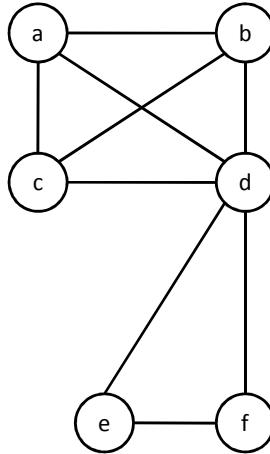
Stap	Topic(s)
1	Implementeer een basisalgoritme waarmee twee afbeeldingen redelijk goed gematcht kunnen worden: <ul style="list-style-type: none"> • Afbeeldingen omzetten naar bijhorende edge maps (eventueel met hulp van een library). Edge maps kunnen hopelijk eenvoudig voorgesteld worden, met polylines bijvoorbeeld. • Afleiden van "objecten" en "relaties" in edge maps • Matching van edge maps via association graphs
2	Lezen over algemene en concrete technieken in image retrieval. Het survey (Ritendra Datta, 2008) komt hiervoor al in aanmerking.
3	Eventueel verdergaand of samengaat met vorige topic: lezen over graph-clustering technieken of graph-databases. Papers die hierover gaan staan opgesomd in het survey (Conte, Foggia, Sansone, & Vento, 2007).
4	Indien er meer efficiëntie nodig is: bekijken van andere maximal clique finding algoritmen, zoals bijvoorbeeld (Sutera, Abu-Khzam, Zhang, Symons, Samatova, & Langston, 2005).
5	Lezen over parameter tuning met least squares indien er veel parameters zijn die moeten ingesteld worden in de eigen implementatie.
6	...

13. Bibliografie

- Blaser, A. D. (2000). *PhD thesis: Sketching Spatial Queries*. Orono, ME: Spatial Information Science and Engineering, University of Maine.
- Bron, C., & Kerbosch, J. (1973). Algorithm 457, Finding All Cliques of an Undirected Graph. *Communications of the ACM, Volume 16, Number 9*, 575-579.
- Burrus, S. C., Gopinath, R. A., & Guo, H. (1998). *Introduction to Wavelets and Wavelet Transforms*. New Jersey: Prentice Hall.
- Cazals, F., & Karande, C. (2008). A note on the problem of reporting maximal cliques. *Theoretical Computer Science, Volume 407*, 564-568.
- Chávez, E. (2001). Searching in Metric Spaces. *ACM Computing Surveys, Volume 33, Issue 3*, 273 - 321.
- Conte, D., Foggia, P., Sansone, C., & Vento, M. (2007). How and Why Pattern Recognition and Computer Vision Applications Use Graphs. In A. Kandel, H. Bunke, & M. Last, *Applied Graph Theory in Computer Vision and Pattern Recognition* (pp. 85-135). Berlin / Heidelberg: Springer.
- Costa, L. d., Rodrigues, F. A., Travieso, G., & Villas Boas, P. R. (2007). Characterization of complex networks: a survey of measurements. *Advances in Physics, Volume 56, Issue 1*, 167 - 242.
- Daubechies, I. (1992). *Ten Lectures on Wavelets*. Philadelphia, Pennsylvania: Society for industrial and applied mathematics.
- Egenhofer, M. J. (1997). Query Processing in Spatial-Query-by-Sketch. *Journal of Visual Languages and Computing, Volume 8, Issue 4*, 403-424.
- Egenhofer, M. J., & Herring, J. R. (1991). *Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases*. Orono, ME: University of Maine, National Center for Geographic Information and Analysis and Department of Surveying Engineering, Department of Computer Science.
- Gottfried, B. (2008). Qualitative similarity measures - The case of two-dimensional outlines. *Computer Vision and Image Understanding, Volume 110, Issue 1*, 117-133.
- Hecht, E. (1998). *Optics, Third Edition*. Addison Wesley.
- Horaud, R., & Skordas, T. (1989). Stereo Correspondence Through Feature Grouping and Maximal Cliques. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 11, No 11*, 1168-1180.
- Kellens, T. (2006). *Database search van tijdreeksen, met toepassing in de firma Medtronic*. Diepenbeek: UHasselt.
- Keogh, E., Wei, L., Xi, X., Lee, S.-H., & Vlachos, M. (2006). LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. *Proceedings of the 32nd international conference on Very large data bases* (pp. 882 - 893). Seoul, Korea: VLDB Endowment.
- Koch, I. (2001). Fundamental Study: Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science, Volume 250*, 1-30.
- Koornwinder, T. H. (1993). *Wavelets: An Elementary Treatment of Theory and Applications*. World Scientific Publishing.

- Levi, G. (1973). A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo, Volume 9, Number 4* , 341-352.
- Loy, G. (2007). *Musimatics, the mathematical foundations of music, Volume 2*. Cambridge, MA: The MIT Press.
- Mallat, S. (1989). A Theory for multiresolution Signal Decomposition: The Wavelet Representation. *IEEE Transactions on pattern analysis and machine intelligence, Vol 11, No 7* , 674-693.
- Mallat, S. (1999). *a Wavelet tour of signal processing*. Academic Press.
- Misiti, M., Misiti, Y., Oppenheim, G., & Poggi, J.-M. (2008). *Wavelet Toolbox™ 4 User's Guide*. Natick, MA: The MathWorks.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Science/Engineering/Math.
- Nedas, K. A. (2006). *PhD dissertation: Semantic Similarity of Spatial Scenes*. Orono, USA: Spatial Information Science and Engineering, University of Maine.
- Nedas, K. A., & Egenhofer, M. J. (2008). Spatial-scene Similarity Queries. *Transactions in GIS, Vol. 12 Issue 6* , 661-681.
- Ricardo Backes, A. e. (2009). A complex network-based approach for boundary shape analysis. *Pattern Recognition, Volume 42, Issue 1* , 54-67.
- Ritendra Datta, D. J. (2008). Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Computing Surveys, Volume 40, Issue 2* .
- Rowland, T. (2009). *Vector Space Tensor Product*. Opgeroepen op 06 11, 2009, van MathWorld--A Wolfram Web Resource: <http://mathworld.wolfram.com/VectorSpaceTensorProduct.html>
- Rudin, W. (1986). *Real and Complex Analysis*. McGraw-Hill Science/Engineering/Math.
- Stollnitz, E., Derose, T., & David, S. (1996). *Wavelets For Computer Graphics*. San Francisco: Morgan Kaufmann Publishers.
- Strang, G. (1989). Wavelets and dilation equations: a brief introduction. *SIAM Review, volume 32, no 4* , 614--627.
- Strang, G., & Nguyen, T. (1996). *Wavelets and Filter Banks*. Wellesley-Cambridge Press.
- Suters, W. H., Abu-Khzam, F. N., Zhang, Y., Symons, C. T., Samatova, N. F., & Langston, M. A. (2005). A New Approach and Faster Exact Methods for the Maximum Common Subgraph Problem. In *Lecture Notes in Computer Science, Computing and Combinatorics, Volume 3595* (pp. 717-727). Berlin / Heidelberg: Springer.
- Van Hoof, J. (2007). *Master Thesis: A Study of Quantitative and Qualitative Methods for Trajectories*. Diepenbeek, Belgium: UHasselt.

14. Appendix



Figuur 64: voorbeeld-graaf

14.1. Uitwerking van het BK-algoritme

Hieronder schrijven we de werking van het algoritme van listing 2 op de graaf in Figuur 64 gedeeltelijk uit. We hebben ons beperkt tot het selecteren van $u_i = a$ in de for-loop van de eerste oproep.

```

 $R = \emptyset, P = \{a, b, c, d, e\}, X = \emptyset$ 
 $u_i = a$ 
 $P = \{b, c, d, e\}$ 
    New call:
     $R = \{a\}, P = \{b, c, d\}, X = \emptyset$ 
     $u_i = b$ 
     $P = \{c, d\}$ 
        New call:
         $R = \{a, b\}, P = \{c, d\}, X = \emptyset$ 
         $u_i = c$ 
         $P = \{d\}$ 
            New call:
             $R = \{a, b, c\}, P = \{d\}, X = \emptyset$ 
             $u_i = d$ 
             $P = \emptyset$ 
                New call:
                 $R = \{a, b, c, d\}, P = \emptyset, X = \emptyset$ 
Output clique  $\{a, b, c, d\}$ 

```

	$X = \{d\}$
	$X = \{c\}$
	$u_i = d$
	$P = \emptyset$
	<p style="text-align: center;">New call:</p> $R = \{a, b, d\}, P = \emptyset, X = \{c\}$
	<p style="text-align: center;">Don't output clique</p> $X = \{c, d\}$
	$X = \{b\}$
	$u_i = c$
	$P = \{d\}$
	<p style="text-align: center;">New call:</p> $R = \{a, c\}, P = \{d\}, X = \{b\}$
	$u_i = d$
	$P = \emptyset$
	<p style="text-align: center;">New call:</p> $R = \{a, c, d\}, P = \emptyset, X = \{b\}$
	<p style="text-align: center;">Don't output clique</p> $X = \{b, d\}$
	$X = \{b, c\}$
	$u_i = d$
	$P = \emptyset$
	<p style="text-align: center;">New call:</p> $R = \{a, d\}, P = \emptyset, X = \{b, c\}$
	<p style="text-align: center;">Don't output clique</p> $X = \{b, c, d\}$
	$X = \{a\}$
	$u_i = b$
	...

DEEL 2

Academiejaar 2009-2010

Inhoudsopgave

1 Visie	5
1.1 Inleiding	5
1.2 Atomaire features	6
1.3 Learning	8
1.3.1 Nesting van feature types	8
1.3.2 Iteratieve learning	9
1.3.3 Feature type database	10
1.3.4 Concrete toepassingen	11
2 Formele concepten	15
2.1 Gelabelde grafen	15
2.2 Induced subgraph	15
2.3 Matchings	16
2.3.1 Complexiteit van Matchings vinden	17
2.4 Maximal common induced subgraph probleem	18
2.4.1 Isomorfe en redundante matchings	18
2.4.2 Subgraaf automorfisme	19
3 Frequent induced subgraph mining	20
3.1 Frequent set intersection mining (FIM)	21
3.1.1 Algoritme	21
3.1.2 Constraints	23
3.2 Frequent subgraph mining	23
3.2.1 Inleiding	23
3.2.2 Algoritme	23
3.2.3 Bespreking	24
3.2.4 Het patroon kan splitsen	26
3.2.5 Omgaan met subgraaf automorfismen	27
3.2.5.1 Actieve matching-uiteinden	27
3.2.5.2 Toepassing	29
3.3 Voorbeeld	29
4 Menselijke perceptie	33
4.1 Visuele functionaliteit in de hersenen	33
4.2 Andere noemenswaardigheden	37
4.3 Visuele informatie beschrijven	39
4.3.1 Feature types	39
4.3.1.1	43

4.3.2	Praktische opmerkingen	43
4.4	V1 gebaseerde piramide	44
4.4.1	Cellen die randen detecteren	44
4.4.2	De piramide	48
4.4.3	Een voorbeeld	49
5	Future work	52

Lijst van figuren

1.1	Schets van de topologie van een feature set, met atomaire features	7
1.2	Relaties ook voorgesteld met knopen.	7
1.3	Moleculair feature type B1	8
1.4	Mogelijke uitwerking van max-pooling in een feature set	11
1.5	De moleculaire features uit Figuur 1.4 zijn hier zelf als knopen afgebeeld.	11
1.6	Een voorbeeld van een feature type database.	12
1.7	Training set van visuele letters “A”	13
1.8	Drie patronen gevonden in de training set van visuele letters “A”	13
1.9	Een nieuw meer high-level patroon, met support 3	13
1.10	Een nieuw meer high-level patroon, met support 2	14
2.1	Redundante matchings	19
2.2	Subgraaf automorfisme	19
3.1	Drie grafen G, H, L en een intersectie-patroon A gevonden in G .	25
3.2	Drie grafen, waarbij het frequente intersectie-patroon in H niet geconnecteerd is.	27
3.3	grafen G, H, L waarbij G een subgraaf automorfisme bevat.	28
3.4	Nesting van matching-uiteinde-groepen	29
3.5	Dit is een abstract voorbeeldje van het (in)actief maken van de uiteinden van zelf-matchings. De grote rechthoek stelt de hele graaf voor en de ovalen stellen de uiteinden voor van zelf-matchings, die zelf in blauwe dunne lijnen zijn weergegeven.	29
3.6	Input-graaf g_1	30
3.7	Input-graaf g_2	31
3.8	Input-graaf g_3	31
3.9	Input-graaf g_4	31
3.10	Gevonden frequent subgraphs (deel 1)	32
3.11	Gevonden frequent subgraphs (deel 2)	32
4.1	Zenuwcellen, allen getuned op een rand/lijn in een bepaalde oriëntatie	35
4.2	Illustratie van inhibition	35
4.3	Illustratie van enhancement	36
4.4	Streepjes en onzekere lijn	43
4.5	De namen van de hokjes in een quad van pixels.	45
4.6	Voorbeelden van randen die je kan detecteren als je slechts 2×2 pixels ter beschikking hebt.	46

4.7	De structuur van de gebruikte neurale netwerken	47
4.8	De gebruiker kan aangeven of een quad voldoet aan het edge type onder beschouwing of niet. Hier wordt concreet de vraag gesteld of de getoonde quad een horizontale rand voorstelt met een heldere bovenkant.	48
4.9	Een editor waarin de gebruiker kan een cell surround model maken dat aangeeft hoe de verschillende edge types elkaar signaal kunnen versterken (facilitation, getoond in groene kleur) of tegenwerken (inhibition, getoond in rode kleur). Elk hokje stelt een aparte cell voor. De getallen in een hokje geven gewoon aan hoeveel opgevulde buur-cellen hij heeft. Voor elk hokje kan apart een dialoog-venster opgeroepen worden om de instellingen te doen. Het dialoog venster in de figuur hoort bij het hokje met een dun blauw randje. Het centrum van het model is getoond in een zwarte stippe lijn.	50
4.10	Originele afbeelding (omgezet naar grijswaarden)	50
4.11	Verwerkte afbeelding	51
4.12	Verwerkte afbeelding	51

Hoofdstuk 1

Visie

In dit hoofdstuk geven we onze visie op een systeem dat gegeven een database van afbeeldingen op een zelfstandige en flexibele manier patronen in deze data kan zoeken, leren, en herkennen. Een dergelijk systeem heeft vele aspecten, waarvan we sommige in dit tweede deel van de thesis al in meer detail hebben kunnen bestuderen terwijl we voor andere de kijtlijnen hebben uitgezet. We schetsen in dit hoofdstuk vooral een aantal kerngedachten die we voor ogen houden als rode draad doorheen de topics en zelfbedachte technieken van dit tweede deel. Ook geven we in dit hoofdstuk waar nodig verwijzingen naar de rest van de tekst.

1.1 Inleiding

In het eerste deel van de literatuurstudie hebben we vele manieren onderzocht om informatie te beschrijven en ook technieken om delen van deze data te modelleren zodat zij met elkaar kunnen vergelijken worden om verbanden te ontdekken. Het zoeken naar vaak voorkomende patronen over een dataset helpt om abstracties te vormen over deze dataset en om deze beter te begrijpen. Deze abstracties kunnen bijvoorbeeld gebruikt worden om patroon-herkenning te doen, omdat een abstractie zich dan richt op de essentiële informatie van een patroon.

Grafen zijn in het algemeen een zeer interessante modelleertechniek omdat deze kunnen gebruikt worden om vele semantische structuren te beschrijven. We hebben in de literatuurstudie onder andere kennis gemaakt met graafmodellering van visuele informatie en technieken zoals de association graph om te zoeken naar (in)exacte overeenkomsten tussen twee grafen. In de literatuur is daarnaast ook het frequent subgraph mining probleem gekend dat zich bezig houdt met het zoeken naar deelgrafen die een voldoende aantal keren voorkomen in een verzameling grafen. Verdergaand op de inhoud van de literatuurstudie hebben we ook zelf technieken bedacht voor dit probleem en sommige daarvan al gedeeltelijk kunnen uittesten. We verwijzen hiervoor door naar hoofdstukken 2 en 3.

In de literatuurstudie is ook aangegeven dat bij opstellen van een graafmodellering van visuele informatie de applicatie beslist welke objecten men extraheert uit de data om voor te stellen met knopen en welke knoop- en boog-labels men gebruikt. Het is niet intuïtief duidelijk hoe men een degelijke en elegante

modellering zou kunnen doen voor bepaalde soorten visuele informatie. In de literatuur zijn ook vele manieren bedacht om bepaalde “features” af te leiden uit een afbeelding en ook technieken om deze met elkaar in verband te brengen over verschillende afbeeldingen heen. Bij deze technieken is het soms nodig om manueel of via machine learning numerieke thresholds te bepalen om de algoritmen te configureren. Dit is niet altijd even eenvoudig en elegant. Bovendien is er niet altijd een duidelijke motivatie waarom de ene techniek beter zou zijn dan een andere.

Daarom hebben we in dit tweede deel ook nieuwe literatuur doorgenomen om kennis te nemen van onderzoek in de neurobiologie en psychologie naar menselijke visuele perceptie. Deze kennis willen we gebruiken om op een meer onderbouwde manier te redeneren over de algoritmen binnen het domein van visuele informatie-verwerking. We bespreken onze bevindingen in paragraaf 4. Door deze kennisse van een ander domein kunnen we in de toekomst ook eenvoudiger nieuwe belangrijke ontdekkingen hierover integreren met onze eigen modelvorming.

1.2 Atomaire features

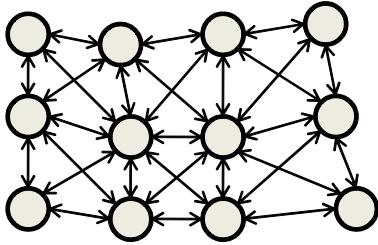
Een “feature” is een kenmerk of eigenschap die men kan toeschrijven aan een stuk data. Een pixel in een afbeelding is in feite een voorbeeld van een zeer elementair “feature type”, omdat dit het kleinste onderdeel is waaruit een afbeelding bestaat. Dit soort elementaire beschrijvende feature types noemen we “atomaire feature types”. Hun concrete instanties in een stuk data noemen we atomaire features. Voor een verdere besprekking over wat features in het algemeen zouden kunnen zijn verwijzen we door naar paragraaf 4.3.

Het is eventueel ook mogelijk om een groepje naburige pixels van eenzelfde intensiteit te beschouwen als een atomair feature. Het komt er op neer dat deze atomaire features op een zeer eenvoudige manier kunnen gevonden worden, zonder dat er extra conceptualisatie over de inhoud van de afbeelding bij komt kijken. Het systeem zal in een eerste fase van elke afbeelding een beschrijving opstellen waarin de aangetroffen atomaire features worden gepresenteerd en hun semantische verbanden. We zullen in hoofdstuk 4 leren dat in de menselijke hersenen er gespecialiseerde cellen bestaan die bij het verwerken van visuele informatie randen in bepaalde oriëntaties kunnen waarnemen. Dit soort “hard-wired” functionaliteit zouden we kunnen beschouwen als detectors voor atomaire features, als algoritmen die steeds op elke input worden uitgevoerd en waarvan de output als niet-doorgrondelijk (dus atomair) wordt beschouwd. De atomaire features vormen in dat opzicht de basisbouwstenen van perceptie.

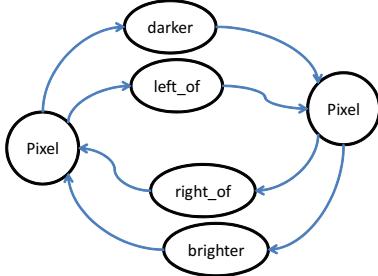
We kunnen de inhoud van een afbeelding beschrijven als een gelabelde (sparse) graaf door voor elk atomair feature een aantal gerichte relaties te definiëren naar een aantal atomaire features in zijn omgeving. Er is dan een label voor de knopen zoals “pixel” of “atomic feature”. Het label van een binaire relatie tussen twee atomaire features x en y zou afkomstig kunnen zijn uit volgende categorieën:

- het beschrijven van de onderlinge positie van x en y
- het vergelijken van attributen van x en y

De binaire relaties tussen de atomaire features kunnen we modelleren met labels zoals “helderder dan”, “donkerder dan”, “links van”, “boven”, ... Merk op dat



Figuur 1.1: Schets van de topologie van een feature set, met atomaire features



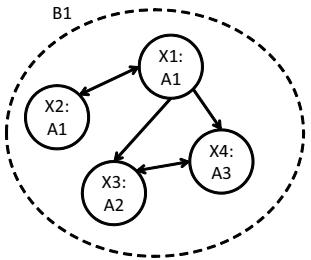
Figuur 1.2: Relaties ook voorgesteld met knopen.

we de vrijheid hebben om meerdere binaire relaties tussen x en y op te stellen. Het is trouwens ook niet verplicht om altijd twee leesrichtingen te hebben voor de binaire relatie tussen twee atomaire features. Het is soms nuttig om slechts één richting te modelleren, bijvoorbeeld van x naar y .

De gelabelde grafen die we als resultaat terugkrijgen van deze semantische beschrijving zullen we ook wel *feature sets* noemen omdat vooral de features zelf hierbij centraal staan. De topologie die we in gedachten hebben voor zulke grafen is sparse, zoals geschetst in Figuur 1.1. Omdat men in principe ook kan kiezen voor andere atomaire features dan de pixels zelf, hebben we de knopen niet geschikt in een rigide rasterform. De pijlen stellen de binaire relaties voor.

Wanneer de knopen in een feature set enkel gebruikt worden om de atomaire features weer te geven kunnen de labels op de bogen complex zijn indien een binaire relatie tussen twee atomaire features veel omschrijving vergt. Hierdoor is het niet altijd eenvoudig om bij het matchen van zulke grafen te zeggen of de labels van twee bogen in voldoende mate overeenkomen. We kunnen de binaire relaties tussen twee atomaire features ook anders modelleren, door een tweede soort knopen te gebruiken. Dit idee is geschetst in Figuur 1.2. Deze variant laat schijnbaar eenvoudiger toe om te redeneren over algoritmen om (dergelijke) grafen te matchen.

Het is in principe mogelijk om meerdere soorten atomaire features waar te nemen in de data, die elk een voldoende onderscheidbaar semantisch feit registreren. Het zou immers bij visuele informatie mogelijk zijn om naast de pixels zelf ook atomaire feature types te onderscheiden die al iets grotere structuren voorstellen zoals bijvoorbeeld randen.



Figuur 1.3: Moleculair feature type B1

1.3 Learning

Van zodra het systeem voor elke afbeelding een eenvoudige graafbeschrijving heeft gemaakt van low-level maar wel belangrijke informatie in een afbeelding, kan het systeem zoeken naar frequente patronen in deze grafen. Dit proberen we in te passen in een omvattend denkkader.

1.3.1 Nesting van feature types

We stellen voor om naast de atomaire feature types ook meer high-level feature types te hebben, de “moleculaire feature types”. Deze nieuwe feature types worden door het systeem zelf gevonden, als patronen in de input. De instanties van moleculaire feature types noemen we moleculaire features. Gezamelijk spreken we over atomaire features en moleculaire features als “features”.

We zullen naar de atomaire feature types verwijzen met de letter A en naar de moleculaire feature types met de letter B . Zij s een instantie van een feature type, dan duiden we met $type(s)$ zijn type aan.

Het kernidee is nu om een soort “recursieve” constructie te maken voor de beschrijving van een moleculair feature type. Een moleculair feature type B definiëren we als een verzameling van *instanties* van andere feature types en verbanden tussen deze instanties. Deze features waaruit B bestaat noemen we de kinderen van B en deze verzameling noteren we met $children(B)$. Zij s een kind van een moleculair feature type, dan duiden we met $parent(s)$ dit moleculaire feature type aan. De types van de kinderen kunnen andere moleculaire feature types zijn of atomaire feature types. Er hoeven in principe niet veel atomaire feature types te bestaan. Het zinvol combineren van atomaire feature types laat immers reeds veel mogelijkheden toe, zeker als moleculaire feature types op hun beurt weer kunnen gecombineerd worden. Een moleculair feature type kunnen we modelleren met een gelabelde graaf $G(B)$.

Figuur 1.3 toont een voorbeeld van een moleculair feature type B1. We zien dat B1 gemodelleerd wordt als een graaf waarbij de knopen een naam en een type hebben. De knoop met naam X1 heeft bijvoorbeeld als type het feature type A1. De pijlen tussen de knopen stellen beschrijvingen voor van de binaire relaties.. Hoe men concreet de binaire relaties modelleert tussen de knopen is eigenlijk in essentie vrij te kiezen.

Een *instantie* van een moleculair feature type definiëren we als een gelabelde graaf H waarbij

- de knopen features zijn en de bogen de concrete binaire relaties beschrijven tussen deze features
- er een bijectie m moet bestaan tussen de kinderen van $G(B)$ en de knopen H zodat een kind van $G(B)$ wordt afgebeeld op een knoop van H met hetzelfde type. Bovendien moeten alle binaire relaties tussen de kinderen van $G(B)$ bewaard blijven. Voor H zelf hoeft dit niet.

Deze vereisten impliceren onder andere dat H evenveel knopen bevat als B kinderen heeft.

1.3.2 Iteratieve learning

Gegeven zijn een aantal atomaire feature types A_1, \dots, A_n en een aantal *feature sets* F_1, \dots, F_m waarin enkel instanties zitten van de gegeven atomaire feature types. Elke feature set F_i is zoals hierboven reeds aangegeven een graaf waarmee de verbanden tussen atomaire features worden uitgedrukt. Mogelijk kunnen hierbij ook eenvoudige hulp-knopen gebruikt worden om de binaire relaties te modelleren.

Zoek binnen deze feature sets naar frequente graaf-patronen. Als we concreet zoeken naar patronen in visuele informatie hebben we meestal op het oog om niet al te kleine overeenkomsten te vinden, maar wel overeenkomsten die visueel zo groot mogelijk ogen.

Elk uniek frequent graaf-patroon kan dan omgezet worden naar een moleculair feature type. Een instantie van zo'n moleculair feature type "komt dan voor" in de grafen waarin dat patroon voorkomt. De aanwezigheid van een moleculair feature type in een feature set kunnen we modelleren door een nieuwe knoop toe te voegen aan deze feature set en de verbanden met de reeds aanwezige atomaire features te beschrijven. Daar gaan we zodadelijk iets dieper op in. Men kan dan vervolgens in deze uitgebreide grafen opnieuw zoeken naar nieuwe graafpatronen, waarin deze keer instanties voorkomen van de pas ontdekte moleculaire feature types. Deze graaf-patronen worden dan op hun beurt omgevormd tot nieuwe moleculaire feature types, en hieruit komt de recursieve notie van de moleculaire features voort. Het is de bedoeling dat we telkens zoeken naar frequente graaf-patronen, deze beschouwen als nieuwe feature types, de feature sets uitbreiden met "instantie-knopen" van deze nieuwe feature types (als dit type daarin voorkomt) en dan dit proces blijven herhalen totdat weinig nieuwe feature types worden gevonden. Moleculaire feature types die we in een later zoekproces vinden dan eerder ontdekte moleculaire feature types bestempelen we als meer "abstract" of van een hoger niveau.

We hebben bovenstaand idee gebaseerd op kennis uit de neurobiologie, waarop we dieper zullen ingaan in hoofdstuk 4. Er bestaat in de menselijke hersenen een gebied genaamd "V1" dat verantwoordelijk is voor de visuele perceptie en ook werkt met een gelijkaardig soort pipeline: uit de low-level prikkels van het oog worden systematisch steeds abstractere voorstellingen geproduceerd die tenslotte visuele herinneringen aanspreken.

Is het in feite mogelijk om in ons geschetste framework herhalend te zoeken naar subgraaf-patronen in de uitgebreide grafen? Vinden we met andere woorden in de tweede frequente subgraaf-zoektocht nog wel nieuwe patronen t.o.v. die van de eerste zoektoch? Dit hangt sterk af van de manier waarop men in een feature set de relaties beschrijft tussen de nieuw-aangetroffen moleculaire

features en de reeds aanwezige features. Als hier voldoende flexibele beschrijvingen mogelijk zijn ontstaan vermoedelijk wel vele nieuwe graaf-patronen. We hebben vanuit de literatuur van de menselijke perceptie onder andere kennis gemaakt met het begrip van *max-pooling* dat de cellen in V1 lijken uit te voeren: de hierarchische processing door lagen van cellen laat schijnbaar enkel de meest “sterke” signalen door. In het artikel [6] wordt beschreven dat de cellen in V1 die in de eerste fasen de visuele input verwerken zich sterker bewust zijn van de ruimtelijke schikking van features, maar dat in “hogere” lagen van V1 enkel nog maar abstracties overblijven die “the big picture” uitdrukken. Dit is ook wat men zou kunnen toepassen bij het beschrijven van de relaties tussen nieuw-aangetroffen moleculaire features t.o.v de reeds aanwezige features, maar ook tussen moleculaire features onderling. Beschouw bijvoorbeeld Figuur 1.4. Daar hebben we twee moleculaire features m_1 en m_2 . Zoals tevoren besproken hebben we een instantie van een moleculair feature gedefinieerd als een graaf van andere features. De knopen (en bijhorende bogen) waaruit een instantie van een moleculair feature bestaat kunnen we de “inhoud” noemen van dat moleculair feature. Voor de eenvoud hebben we in de figuur de relaties binnen de inhoud van elk moleculair feature niet getekend. We kunnen nu alle relaties beschouwen die de knopen van m_1 hebben naar de knopen in m_2 . Voor de eenvoud hebben we deze beschrijvingen genoteerd in één boog-label. We kunnen nu alle labels verzamelen die voldoende sterk aanwezig zijn in deze collectie van bogen. We kunnen bijvoorbeeld stellen dat een label bij minstens de helft van deze bogen moet voorkomen. In dit geval levert ons dat de labels “left_of”, “below” en “darker” op. Het is ook mogelijk om eerst te constateren of het aantal bogen dat vanuit m_1 vertrekt t.o.v. het totaal aantal mogelijke bogen wel voldoende hoog is en enkel indien er genoeg bogen zijn de relatie te beschrijven. We noemen dan de gerichte relatie van m_1 naar m_2 significant. Als we enkel de m_1 en m_2 in dit voorbeeld tenslotte voorstellen als hun abstracte knopen (in een feature set) krijgen we Figuur 1.5.

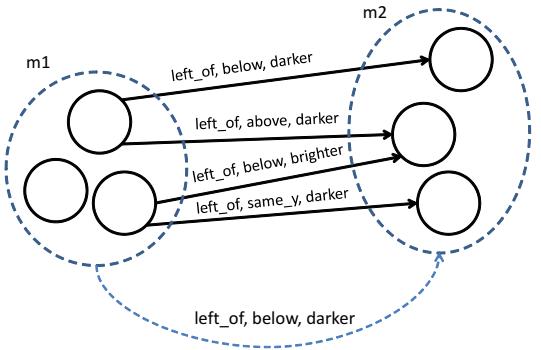
Dit alles kan men ook doen voor de richting van m_2 naar m_1 . Hetzelfde principe kan men ook toepassen voor het beschrijven van een moleculair feature naar een feature van een lager niveau.

Het is ook mogelijk dat geen voldoende frequente labels kunnen gevonden worden tussen twee moleculaire features, en de gerichte relatie in kwestie feitelijk toch significant is. In dat geval zou men een nieuw label “costimulated with” kunnen gebruiken om toch aan te geven dat er een sterk verband is tussen beide features.

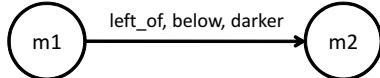
1.3.3 Feature type database

Initiëel kent ons systeem enkel de atomaire feature types. Maar door learning is het mogelijk dat er nog een aantal moleculaire feature types worden afgeleid uit de dataset. We kunnen een gerichte graaf opstellen als volgt: trek vanuit elke kind-knoop van een moleculair feature type een pijl naar het bijhorende feature type t . Deze pijl drukt expliciet uit dat het type van dit kind t is. We noemen de resulterende graaf de *feature type database* \mathcal{D} .

Zij t een feature type, dan noteren we met $incoming(t)$ de verzameling van alle kinderen van moleculaire feature types die t als type hebben. We veronderstellen dat de feature type database *geen cycles* bevat. Dit kan ook niet voorkomen door de manier waarop we de learning van moleculaire fea-



Figuur 1.4: Mogelijke uitwerking van max-pooling in een feature set



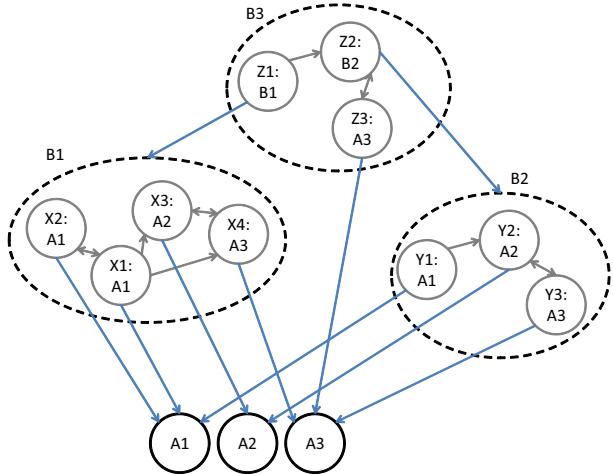
Figuur 1.5: De moleculaire features uit Figuur 1.4 zijn hier zelf als knopen afgebeeld.

tures hierboven hebben beschreven. In Figuur 1.6 is een eenvoudig voorbeeldje weergegeven. Daar geldt $incoming(A1) = \{X1, X2, Y1\}$, $incoming(A2) = \{X3, Y2\}$, $incoming(A3) = \{X4, Y3, Z3\}$, $incoming(B1) = \{Z1\}$, $incoming(B2) = \{Z2\}$, $incoming(B3) = \emptyset$.

Wanneer voor een gegeven database van afbeeldingen bijvoorbeeld de feature type database is opgesteld en we dan een nieuwe afbeelding I krijgen aangeboden is het mogelijk om in deze nieuwe afbeelding I instanties van alle feature types in de feature type database waar te nemen. Omdat de moleculaire feature types soms afhangen van elkaar is het nodig om de features te detecteren in fasen. Men zet I eerst om naar een feature set F met enkel atomaire features in. Vervolgens kunnen we in de feature type database opzoeken welke kinderen van moleculaire feature types B_i deze atomaire feature types als type hebben. De atomaire features in F kunnen we dan rechtstreeks gebruiken als “ingrediënten” voor de gebruikmakende moleculaire feature types B_i . Natuurlijk moeten we daarbij wel controleren of de verbanden die beschreven zijn tussen de kinderen van een molecuair feature ook effectief voldaan zijn in de feature set F . Van zodra instanties van moleculaire feature types B_i gevonden zijn kunnen we deze toevoegen aan F . We herhalen recursief de feature detector procedure door in de feature type database op te zoeken welke kinderen van welke moleculaire feature types B_j gebruik maken van het type B_i . Vervolgens kunnen we met de instanties van het type B_i en ook de andere aanwezige features dan instanties van het type B_j af te leiden. Dit herhalen we totdat we geen nieuwe features meer kunnen afleiden.

1.3.4 Concrete toepassingen

Het automatisch aanleren van de moleculaire feature types kent mogelijk vele toepassingen. We kunnen bijvoorbeeld afbeeldingen clusteren op basis van welke



Figuur 1.6: Een voorbeeld van een feature type database.

feature types zij instanties hebben. Afbeeldingen die instanties van meer abstracte feature types bevatten hebben een hogere kans om gelijkaardig te zijn.

Daarnaast zou het ook mogelijk kunnen zijn om voor een bepaalde visuele 2D entiteit zoals ‘‘letter A’’, ‘‘Kruis van Lotharingen’’, enz een aparte verzameling afbeeldingen op te stellen waarbinnen dan gezocht wordt naar moleculaire beschrijvende feature types. Deze afbeeldingen noemen we dan de training set. Het is de bedoeling dat deze afbeeldingen voldoende visueel gelijkaardig zijn en toch verschillend genoeg zijn zodat het algoritme moleculaire feature types kan afleiden die de 2D entiteit in kwestie voldoende algemeen kunnen beschrijven en toelaten om deze entiteit dan in nieuwe afbeeldingen te herkennen. We geven nu een meer concrete illustratie van wat we hier mee bedoelen. Deze illustratie beeldt uit wat ons systeem in principe zou kunnen doen, maar we hebben dit voorbeeld met de hand gemaakt omdat we dit systeem nog niet ter beschikking hebben. Beschouw Figuur 1.7. Dit voorbeeld zou een kleine training set kunnen zijn voor de visuele entiteit (drukletter) ‘‘A’’. Veronderstel dat de afbeeldingen uit voldoende pixels bestaan om zinvolle feature sets op te stellen, met de pixels als atomaire features. Als we zouden zoeken naar frequente graaf-patronen die in minstens de helft van de feature sets voorkomen, krijgen we bijvoorbeeld het blauwe, oranje en groene patroon getoond in Figuur 1.8. Deze patronen zetten we om naar moleculaire featur types. In de eerste en derde ‘‘A’’ komen dan instanties van de drie nieuwe moleculaire feature types voor en in de tweede ‘‘A’’ slechts twee instanties. We voegen deze instanties formeel toe als knopen aan de feature sets van de letters ‘‘A’’. We beschrijven ook de relaties van deze moleculaire features t.o.v. elkaar en t.o.v. de reeds aanwezige atomaire features. Als we dan in deze uitgebreide grafen op zoek gaan naar nieuwe patronen krijgen we bijvoorbeeld het rode patroon getoond in Figuur 1.9 en het gele patroon getoond in Figuur 1.10. We zien dat in de tweede letter ‘‘A’’ het rode patroon visueel breder is dan in de andere letters. Dat komt omdat de afstand tussen het onderliggende blauwe en oranje feature groter is dan in de andere letters, maar toch voldoende klein opdat we de binaire relatie ertussen zinvol kunnen beschrijven. Ondanks dit kleine visuele *verschil* wordt dit rode patroon in de



Figuur 1.7: Training set van visuele letters “A”



Figuur 1.8: Drie patronen gevonden in de training set van visuele letters “A”

tweede letter “A” als gelijk beschouwd aan de rode patronen in de eerste en derde letters. Hier zit volgens ons een mogelijke sterkte van het idee, namelijk dat door op een flexibele manier relaties tussen moleculaire features te beschrijven deze beschrijvingen in verschillende feature sets toch kunnen overeenkomen om aanleiding te geven tot een nieuw frequent patroon. Het geschatste idee van de max-pooling hierboven kan hier eventueel voor zorgen, omdat daar enkel de labels worden onthouden die voldoende sterk aanwezig zijn ontstaat een abstractere representatie van de binaire relatie tussen twee moleculaire features die een hogere kans heeft om vaker voor te komen in verschillende feature sets. In elke individuele letter zijn dan de concrete schikkingen van de features in feite lichtjes verschillend, maar het frequente patroon is dan een abstractie over de letters heen. Eenzelfde redenering kan men ook maken voor het gele patroon, waarvan de instantie in de eerste letter niet honderd percent gelijk is aan de instantie in derde letter.



Figuur 1.9: Een nieuw meer high-level patroon, met support 3



Figuur 1.10: Een nieuw meer high-level patroon, met support 2

Hoofdstuk 2

Formele concepten

We herhalen hier belangrijke begrippen en introduceren ook enkele nieuwe t.o.v. de literatuurstudie.

2.1 Gelabelde grafen

Een gelabelde graaf G is een koppel $(V(G), E(G))$ met $V(G)$ de verzameling knopen en $E(G)$ de verzameling bogen. Een boog is een koppel $(v, w) \in V(G) \times V(G)$. Wanneer $v \neq w$ beschouwen we de boog (v, w) verschillend van de boog (w, v) . We beschouwen dus in feite gerichte grafen. Het is niet verplicht dat er tussen elk paar knopen een boog aanwezig is. We noemen een koppel knopen “incident” als er een boog tussen bestaat, in eender welke richting. Een graaf is geconnecteerd als er een sequentie van bogen bestaat (de richting maakt niet uit) die elk paar knopen in de graaf verbindt.

Zij Σ een verzameling labels. We veronderstellen dat er een labeling functie $\lambda : V(G) \cup E(G) \rightarrow \Sigma$ bestaat die aan elke knoop en boog in de graaf een label toekent.

Kleine opmerking: een graaf zonder labels op zijn knopen en bogen is eigenlijk een speciaal soort gelabelde graaf, waarbij we een soort *null* label toekennen aan de knopen en bogen. We veronderstellen daarom dat elke graaf die we hieronder zullen beschouwen gelabeld is. Bovendien kan men ongerichte grafen beschouwen als speciale gevallen van gerichte grafen.

2.2 Induced subgraph

Zij G en H twee grafen. We noemen G een induced subgraph van H , genoteerd $G \subseteq H$ wanneer

$$V(G) \subseteq V(H), E(G) \subseteq E(H)$$

$$\forall v, w \in V(G) : (v, w) \in E(G) \Leftrightarrow (v, w) \in E(H)$$

2.3 Matchings

Zij G en H twee grafen waarbij $V(G) \cap V(H) = \emptyset \wedge E(G) \cap E(H) = \emptyset$. Een *matching* van G in H definiëren we als een partiële injectie $r : V(G) \rightarrow V(H)$ zodat $\forall v \in \text{dom}(r) : \lambda(r(v)) = \lambda(v)$.

Herinner, een partiële injectie is feitelijk een verzameling koppels van de vorm $(v, r(v))$. Een matching r van G in H geeft aanleiding tot een matching r^{-1} van H in G .

We zeggen dat een matching $r : V(G) \rightarrow V(H)$ *bevat* is in matching $s : V(G) \rightarrow V(H)$ wanneer $\text{dom}(r) \subseteq \text{dom}(s) \wedge \forall w \in \text{dom}(r) : r(w) = s(w)$.

Beschouw een matching $r : V(G) \rightarrow V(H)$, dan noemen we $\text{dom}(r)$ en $\text{img}(r)$ de *matching-uiteinden*: $\text{dom}(r)$ is het matching-uiteinde dat in $V(G)$ valt en $\text{img}(r)$ is het matching-uiteinde dat in $V(H)$ valt.

Het effect van een matching kunnen we ook definiëren op de gelabelde bogen. Een boog $e = (v, w) \in E(G)$ wordt *bewaard* door een matching r als $\lambda(r(e)) = \lambda(e) \wedge (r(v), r(w)) \in H$. Noteer

$$\text{bewaard}(r) := \{e \in E(G) \mid e \text{ is een gelabelde boog, bewaard door } r\}$$

We kunnen nu voorwaarden verzinnen waaraan een matching moet voldoen. Bijvoorbeeld, voor de matching $r : V(G) \rightarrow V(H)$ kunnen we eisen dat voor elk paar incidente knopen alle bogen bewaard worden (maximaal twee). Een ander idee zou kunnen zijn dat we gewichten toekennen aan de bogen en eisen dat r voor elk paar incidente knopen bogen bewaart met hun gewicht boven een bepaalde threshold. Er zijn nog tal van variaties mogelijk, voor zowel richting $V(G) \rightarrow V(H)$ als richting $V(H) \rightarrow V(G)$ afzonderlijk. Een matching die voldoet aan dergelijke voorwaarden noemen we *aanvaardbaar* en de voorwaarden zelf kunnen we *aanvaardbaarheidsvoorwaarden* noemen. Een matching $r : V(G) \rightarrow V(H)$ kan aanvaardbaar zijn, maar dat impliceert niet dat matching r^{-1} aanvaardbaar is en omgekeerd.

We kunnen deze aanvaardbaarheidsvoorwaarden uitdrukken met een predicataat α dat twee verzamelingen bogen neemt en *true* of *false* teruggeeft. We zeggen dat een matching $r : V(G) \rightarrow V(H)$ “voldoet” aan α als $\forall v, w \in \text{dom}(r)$ geldt

$$\alpha(\{(v, w) \mid (v, w) \in E(G)\}, \{(r(v), r(w)) \mid (r(v), r(w)) \in E(H)\}) = \text{true}$$

De α kan *symmetrisch* zijn, wat betekent dat de voorwaarde om bogen te bewaren van G in H dezelfde is als de voorwaarde om bogen te bewaren van H in G en dat α deze tegelijk controleert. Bij zulke symmetrische α zullen beide richtingen van de matching ofwel samen voldoen aan de voorwaarden ofwel niet.

Een voorbeeld van een niet-symmetrische α :

$$\alpha(E_1, E_2) := \exists e_1 : (E_1 = \{e_1\}) \Rightarrow \exists e_2 : (E_2 = \{e_2\} \wedge \lambda(e_1) = \lambda(e_2))$$

Een voorbeeld van een symmetrische α :

$$\alpha(E_1, E_2) := \{\lambda(e) \mid e \in E_1\} = \{\lambda(e) \mid e \in E_2\}$$

Een matching die voldoet aan een bepaalde α noemen we een α -matching.

2.3.1 Complexiteit van Matchings vinden

Als we een bepaalde arbitraire aanvaardbaarheidsvoorwaarde zouden vastleggen waaraan matchings moeten voldoen, definiëren we het beslissingsprobleem van het zoeken naar matchings als volgt:

Beslissingsprobleem α -matchings
Gegeven: twee gelabelde grafen G, H en $k \in \mathbb{N}$
Gevraagd: bestaat er een α -matching van G naar H die minstens gedefinieerd is op k knopen van G (en H)?

De aanvaardbaarheidsvoorwaarde behoort dus niet tot de invoer en is altijd hetzelfde. Voor elke α krijgen we een ander beslissingsprobleem. Weten voor welke soorten α 's dit beslissingsprobleem NP-compleet is, is nog een open vraag.

Beschouw de aanvaardbaarheidsvoorwaarde $\alpha_{induced}$:

$$\alpha_{induced}(E_1, E_2) := \{\lambda(e) \mid e \in E_1\} = \{\lambda(e) \mid e \in E_2\}$$

Deze aanvaardbaarheidsvoorwaarde stelt voor een matching van twee grafen dat tussen elk paar knopen alle bogen bewaard worden in beide richtingen. Dit is ook de symmetrische voorbeeld- α van de vorige paragraaf.

Eigenschap: het $\alpha_{induced}$ -matchings probleem is NP-compleet. Om te zien dat dit probleem in NP zit kunnen we op een niet-deterministische manier een matching raden en controleren of de bogen paarsgewijs aan $\alpha_{induced}$ voldoen. We kunnen vervolgens het Induced Subgraph Isomorphism probleem (NP-compleet) reduceren naar dit specifieke $\alpha_{induced}$ -matching probleem.

Induced Subgraph Isomorphism → $\alpha_{induced}$ -matchings
Gegeven: twee gerichte grafen G, H met het aantal knopen in G gelijk aan k . Een ongerichte graaf is eigenlijk een gerichte graaf met symmetrische bogen, dus daar moeten we geen aparte geval voor beschouwen. Reductie: Zij l een label. Maak gelabelde graaf G' door elke knoop en boog in G hetzelfde label l toe te kennen. Doe dit ook voor H en bekom gelabelde graaf H' . Merk op dat het aantal knopen en bogen in beide grafen dus niet verandert. Deze omzetting kan gebeuren in een tijd die polynomiaal is in functie van de grootte van G en H . Geef nu als antwoord $\alpha_{induced}$ -matchings(G', H', k).

We besluiten dat het α -matchings probleem voor sommige α NP-compleet kan zijn.

2.4 Maximal common induced subgraph probleem

In de literatuurstudie hebben we in hoofdstuk 10 het maximal common induced subgraph (MCIS) probleem genoemd als een exacte methode om overeenkomsten te vinden tussen twee grafen. De association graph is een mogelijke techniek om deze overeenkomsten te vinden. We herhalen deze techniek, in het licht van de nieuw-introduceerde concepten van matchings.

We veronderstellen in de context van dit probleem dat de aanvaardbaarheidsvoorraarde op matchings steeds de $\alpha_{induced}$ van vorige paragraaf is.

Beschouw twee gelabelde grafen G en H . De bijhorende association graph $A = (V_A, E_A)$ is ongericht. De association nodes V_A kunnen gemaakt worden onafhankelijk van de aanvaardbaarheidsvoorraarden:

$$V_A = \{(v, w) \mid v \in V(G), w \in V(H), \lambda(v) = \lambda(w)\}$$

Zij (a, x) en (b, y) twee association nodes. We trekken een boog wanneer

$$\begin{aligned} &\alpha_{induced}(\{(a, b) \in E(G)\}, \{(x, y) \in E(H)\}) \wedge \\ &\alpha_{induced}(\{(b, a) \in E(G)\}, \{(y, x) \in E(H)\}) \end{aligned}$$

We zoeken naar maximal cliques in de association graph. Dit kan gebeuren met de Bron-Kerbosch heuristiek van hoofdstuk 11 uit de literatuurstudie.

Elke clique, en dus elke maximal clique, komt overeen met een $\alpha_{induced}$ -matching $r : V(G) \rightarrow V(H)$. Beschouw een maximal clique C . Voor elke association node $(a, x) \in C$ stellen we $r(a) = x$. Door de manier waarop $\alpha_{induced}$ gedefinieerd is hebben we ook precies de voorwaarden van een induced subgraph isomorphism gecontroleerd: als graaf G een boog heeft tussen twee knopen van $dom(r)$, dan heeft graaf H ook een boog (met hetzelfde label) tussen de projecties van die knopen. Dit geldt ook omgekeerd omdat de $\alpha_{induced}$ symmetrisch is. Een $\alpha_{induced}$ -matching r van G naar H geeft dus aanleiding tot een $\alpha_{induced}$ -matching r^{-1} van H naar G .

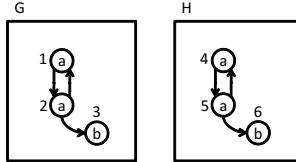
Voor de andere richting, beschouw een $\alpha_{induced}$ -matching $r : V(G) \rightarrow V(H)$. We komen elk koppel $(a, r(a)) \in r$ tegen als knoop in de association graph. Zij $(a, r(a)) \in r$ en $(b, r(b)) \in r$. Omdat de bogen tussen knopen a en b en tussen $r(a)$ en $r(b)$ voldoen aan de $\alpha_{induced}$ -voorraarde, zal er in de association graph een boog bestaan tussen de association nodes $(a, r(a))$ en $(b, r(b))$. Alle koppels van r zitten daarom samen in een clique.

Elke maximal clique in de association graph komt overeen met een soort “maximale matching”. Een strikte deelclique van zo’n clique zou geen extra nuttige informatie opleveren t.o.v. de omvattende maximal clique.

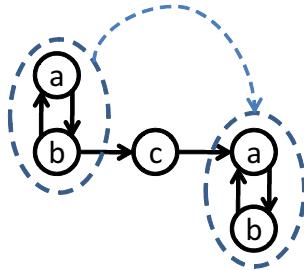
We besluiten dat we de maximale overeenkomsten tussen twee grafen kunnen uitdrukken via matchings.

2.4.1 Isomorfe en redundante matchings

Beschouw twee matchings $r : V(G) \rightarrow V(H)$ en $r' : V(G) \rightarrow V(H)$ bij het MCIS probleem. Het is mogelijk dat $dom(r) = dom(r')$ en dat $img(r) = img(r')$. We noemen dit soort matchings *isomorf* omdat de grafen $dom(r)$, $dom(r')$, $img(r)$ en $img(r')$ allemaal isomorf zijn aan elkaar. We willen zulke matchings niet



Figuur 2.1: Redundante matchings



Figuur 2.2: Subgraaf automorfisme

onderscheiden omdat zij in essentie geen overeenkomst tussen de twee grafen aangeven. Het is mogelijk dat ook meer dan twee matchings isomorf zijn. We onthouden uit elke groep isomorfe matchings slechts één matching.

Bovendien moeten we goed opletten met het volgende. Noteer met $r'(V)$ het beeld van enkel de deelverzameling V , waarbij $V \subseteq \text{dom}(r')$. Wanneer $\text{dom}(r) \subsetneq \text{dom}(r') \wedge r'(\text{dom}(r)) = \text{img}(r)$ noemen we r redundant. We onthouden dan niet r , maar r' als die tenminste zelf niet redundant is. Beschouw bijvoorbeeld Figuur 2.1. De matching $\{(1, 5); (2, 4)\}$ is redundant t.o.v. de matching $\{(1, 4); (2, 5); (3, 6)\}$.

2.4.2 Subgraaf automorfisme

Beschouw een graaf G . Het is mogelijk om het MCIS probleem op te lossen tussen G en G zelf. De resulterende matchings noemen we *zelf-matchings*. Onder de zelf-matchings hebben we onder andere de identiteit als mogelijke matching en de bijhorende isomorfe matchings, maar misschien ook “kleinere” matchings $r : V(G) \rightarrow V(G)$ waarbij $\text{dom}(r) \subsetneq V(G)$. Elk zulke matching wijst erop dat een strikte deelgraaf $G' \subsetneq G$ isomorf is aan een andere strikte deelgraaf $G'' \subsetneq G$.

In Figuur 2.2 is een voorbeeld geschetst van een graaf waarbij subgraaf automorfismen voorkomen (aangegeven in blauwe stippeellijnen).

We zullen de begrippen van isomorfe, redundante en zelf-matchings gebruiken in hoofdstuk 3.

Hoofdstuk 3

Frequent induced subgraph mining

Het zoeken naar frequent (induced) subgraphs in een graaf-database is een gekend informatica probleem. Zij \mathcal{G} een verzameling (database) van grafen. Zij G een graaf. De support van G in \mathcal{G} , genoteerd $sup(G, \mathcal{G})$ is

$$\Sigma_{H \in \mathcal{G}} s(G, H)$$

waarbij $s(G, H)$ gelijk is aan 1 wanneer $G \subseteq H$ en anders 0.

Zij σ een threshold, de *minsup* threshold genaamd. We zeggen dat een graaf G een *frequent induced subgraph* is wanneer $sup(G, \mathcal{G}) \geq \sigma$. Als de database \mathcal{G} duidelijk is vanuit de context schrijven we hem niet explicet meer. Het woord “induced” zullen we vanaf nu ook veronderstellen en niet meer vermelden.

We willen enkel frequent subgraphs bekomen die geconnecteerd zijn, omdat deze het eenvoudigst te begrijpen zijn en omdat dit de mogelijke outputs helpt reduceren. We hebben een eigen algoritme bedacht om te zoeken naar dit soort frequent subgraphs in een verzameling grafen. Een korte literatuurstudie naar de gekende algoritmen voor dit probleem lijkt een aantal tekortkomingen aan het licht te brengen [17, 1]. De meeste algoritmen proberen grafen te “genereren” die mogelijk aanwezig zijn in de graaf-database \mathcal{G} en tellen daarna de support ervan. Het nadeel hiervan is echter dat bij een grote verzameling van mogelijke labels veel grafen tevergeefs zullen gegenereerd worden.

Onze bijdrage is het verkennen van een andere aanpak voor het vinden van frequent subgraphs. Ons algoritme probeert patronen te zoeken door zeer “dicht” bij de feitelijke data zelf te blijven, zonder alle mogelijke grafen te beschouwen en we hopen dat dit daarom bij sommige toepassingen een snelheidsvoordeel kan betekenen. Bovendien zijn er in ons algoritme duidelijke mogelijkheden tot parallelisatie.

We voegen ook een extra parameter z toe, de *minsize* threshold. We willen enkel frequente subgrafen die bestaan uit minstens z knopen.

3.1 Frequent set intersection mining (FIM)

Het volgende sub-probleem hebben we zelf bedacht. Er is wel een mogelijk verband met het minen naar (grote) itemsets in een verzameling item-transacties, een probleem dat welbekend is in de datamining literatuur. Dit verband hebben we nog niet concreet onderzocht.

Gegeven een verzameling knopen (of items) N en een aantal deelverzamelingen hiervan $\mathcal{S} \subseteq \{s | s \subseteq N\}$. Zij $\mathcal{I} \subseteq \mathcal{S}$ een niet-lege doorsnede van een aantal verzamelingen in \mathcal{S} . We definiëren de support van \mathcal{I} , genoteerd $sup(\mathcal{I})$ als $|\mathcal{I}|$. De knopen waaruit de doorsnede bestaat, genoteerd $\mathcal{I}(N)$, definiëren we als $\bigcap \mathcal{I}$. In dat opzicht is een doorsnede van meerdere verzamelingen een soort filter op een aantal knopen uit N . De grootte van de doorsnede, genoteerd $size(\mathcal{I})$ definiëren we als $|\mathcal{I}(N)|$. We stellen dat $\emptyset(N) = N$.

Zij N een verzameling knopen en zij $s \in \mathcal{S}$ een andere verzameling, dan zeggen we dat $N \cap s$ de “projectie” is van s binnen N .

Beschouw twee parameters $\sigma, z \in \mathbb{N}$. We noemen σ de *minsup* threshold en z de *minsize* threshold.

Probleemstelling: vind alle doorsneden \mathcal{I} waarvoor $sup(\mathcal{I}) \geq \sigma$ en $size(\mathcal{I}) \geq z$.

Beschouw twee doorsneden \mathcal{I}_1 en \mathcal{I}_2 . We noemen \mathcal{I}_2 redundant als $\mathcal{I}_2(N) \subseteq \mathcal{I}_1(N)$ en $sup(\mathcal{I}_2) \leq sup(\mathcal{I}_1)$. De uitdaging bestaat erin om een algoritme te ontwerpen dat geen redundante doorsneden vindt.

3.1.1 Algoritme

Een zelf-bedacht algoritme voor het frequent set mining probleem is gegeven in Algoritme 3.1.

We breiden hier recursief een doorsnede \mathcal{F} uit. Deze doorsnede noteren we met \mathcal{F} omdat deze een soort filter vormt op de knopen. De andere verzameling \mathcal{P} is de verzameling van afgewerkte (“processed”) verzamelingen. De knopen gefilterd door de filter noteren we met N' . Als we merken aan het begin van de recursieve oproep dat de hoeveelheid gefilterde knopen onder de minimum-threshold z ligt kunnen we de oproep vroegtijdig afbreken omdat door het uitbreiden van de filter de doorsnede enkel maar kleiner zal worden.

Bij elk oproep beschouwen we andere verzamelingen die we nog kunnen toevoegen aan de filter. Deze toevoegingen zitten in de verzameling E , de “extensions” verzameling. We mogen natuurlijk enkel extensions beschouwen naar verzamelingen die effectief overlappen met de gefilterde knopen. Bij elke oproep mogen we niet meer de verzamelingen beschouwen die reeds in de filter zitten of die reeds zijn afgewerkt, want anders krijgen we dubbele meldingen van dezelfde doorsnede. Daarna verwijderen we alle verzamelingen waarvan de projectie binnen de afgewerkte knopen bevat zit in de projectie van een afgewerkte verzameling. Merk op dat we hiertoe niet de unie van de afgewerkte verzamelingen beschouwen. Dit zou overmatig restrictief zijn omdat het mogelijk is dat een extension nooit volledig wordt bedekt door een afgewerkte verzameling en daardoor ook nog niet volledig is beschouwd op zichzelf. Tenslotte breiden we ook enkel uit naar verzamelingen die niet strikt bevat zijn in andere verzamelingen. Dit noemen we het zoeken naar de maximale extensions, wat gebeurt via het topologisch sorteren van de extensions verzameling.

Algorithm 3.1 Algoritme voor frequent set intersection mining

```
function initial( $\mathcal{S}, \sigma, z$ )
    recursion( $\emptyset, \emptyset$ )  
  
function void recursion( $\mathcal{F}, \mathcal{P}$ ) {
     $N' = \mathcal{F}(N)$ 
    if  $|N'| < z$ 
        return  
  

     $E = \{s \in \mathcal{S} \mid \exists n \in N' : n \in s\}$ 
     $E = E \setminus \mathcal{F}$ 
     $E = E \setminus \mathcal{P}$ 
     $E = E \setminus \{s \in E \mid \exists p \in \mathcal{P} : (s \cap N') \subseteq (p \cap N')\}$ 
     $E = E \setminus \{s \in E \mid \exists t \in E : (s \cap N') \subsetneq (t \cap N')\}$   
  

     $E_c = E$ 
    for all  $e \in E_c$  {
         $A = \{d \in E_c \mid (e \cap N') = (d \cap N')\} \cup \{e\}$ 
         $E = E_c \setminus A$ 
        recur( $\mathcal{F} \cup A, \mathcal{P}$ )
         $\mathcal{P} = \mathcal{P} \cup A$ 
    }
     $\mathcal{P} = \mathcal{P} \setminus E$   
  

    if  $\text{size}(N') \geq z \wedge \text{sup}(\mathcal{F}) \geq \sigma$ 
        output( $\mathcal{F}$ )
    }
}
```

Vervolgens maken we een kopie E_c van E waaruit we ook elementen mogen weghalen. Voor elke extension $e \in E_c$ beschouwen we alle *projectie-equivalente* extenties. Dit zijn verzamelingen d waarvan de projectie binnen de gefilterde knopen precies gelijk is aan de projectie van e binnen de gefilterde knopen. Het heeft geen zin deze verzamelingen afzonderlijk toe te voegen aan de filter omdat daardoor mogelijk dubbele output-doorsneden ontstaan met eenzelfde support. Zulke projectie-equivalente verzamelingen voegen we daarom gelijktijdig toe aan de filter. Voor elke filter-uitbreiding doen we een recursieve oproep.

Na de oproep van de kinderen halen we de extensions e terug uit de verzameling \mathcal{P} omdat deze misschien slechts ten dele aanwezig waren in N' . Als we dan een recursie-niveau terug springen is e misschien veel groter en nog niet afgewerkt in zijn geheel. Binnen een recursieve oproep met een filter zijn natuurlijk de projecties van de verzamelingen kleiner en mogen we hen toevoegen aan \mathcal{P} omdat dit een beperkter subprobleem is.

3.1.2 Constraints

We kunnen het vorige algoritme ook uitbreiden zodat er rekening kan gehouden worden met user constraints. Concreet hebben we in gedachten dat de gebruiker een aantal extra verzamelingen kan specificeren waarbinnen geen frequente doorsneden mogen gevonden worden. Deze verzamelingen noemen we “constraints”.

Men kan de constraints modelleren door deze constraints initieel al in de verzameling \mathcal{P} te plaatsen. Buiten de parameters \mathcal{S}, σ en z ontvangt de frequent set mining routine dan ook nog deze initiële constraint-set \mathcal{P} . Deze constraints worden door de recursieve oproepen van het algoritme nooit weggegooid.

3.2 Frequent subgraph mining

3.2.1 Inleiding

In deze paragraaf geven we een beschrijving van een zelfbedacht algoritme om te zoeken naar frequent subgraphs.

Zij σ de minsup threshold en z de minsize threshold. Zij \mathcal{G} de graaf-database met n grafen. Veronderstel voorlopig dat er geen subgraaf automorfismen aanwezig zijn in een graaf van \mathcal{G} .

Het kernidee van het algoritme is in het kort als volgt. We berekenen tussen elk paar grafen in de database de paarsgewijze matchings. Vervolgens gaan we in elke graaf de uiteinden van deze matchings beschouwen als verzamelingen waartussen we de frequent set intersections kunnen zoeken. Hiervoor wordt dan bijvoorbeeld het algoritme van paragraaf 3.1.1 gebruikt. Elk zulke frequent set intersection wordt dan beschouwd als een frequent subgraph. We moeten hierbij echter ook maatregelen nemen om zoveel mogelijk te vermijden dat een frequent subgraph meermaals wordt gemeld.

3.2.2 Algoritme

In een eerste fase lossen we voor elk paar $G, H \in \mathcal{G}$ het MCIS probleem op, zoals aangegeven in paragraaf 2.4. Het resultaat is een verzameling matchings, waarbij tussen elk paar grafen mogelijk meerdere matchings gevonden worden.

We beschouwen enkel matchings die gedefinieerd zijn op minstens z knopen en waarbij de uiteinden geconnecteerd zijn. We beschouwen ook voor elk paar grafen slechts matchings in één richting, want door de symmetrische aanvaardbaarheidsvoorwaarden krijgen we anders redundante matchings. Tenslotte, we nemen van elke groep isomorfe matchings slechts één matching en beschouwen geen redundante matchings.

Het berekenen van alle zulke paarsgewijze matchings kan voor elk paar grafen in principe in parallel gebeuren.

Zij μ een matching-uiteinde, dan noteren we met $m(\mu)$ het andere uiteinde. Noteer met \mathcal{U} de verzameling van alle mogelijke matching uiteinden. Dan noteren we met $\mathcal{U}(G) = \{\mu \in \mathcal{U} \mid \mu \subseteq V(G)\}$, m.a.w. de matching-uiteinden die vallen binnen graaf G . We zeggen dat een graaf G een matching-uiteinde μ “heeft” in graaf H wanneer $m(\mu) \in \mathcal{U}(H)$ en $\mu \in \mathcal{U}(G)$. Het is mogelijk dat zo’n G meerdere matching-uiteinden heeft in H . Al zulke μ noteren we met $\mathcal{U}(G, H)$ en lezen we als “de matching-uiteinden van G in H ”.

Beschouw een willekeurige volgorde op de grafen in \mathcal{G} : G_1, \dots, G_n . Beschouw G_i met $i = 1 \dots n$. Maak twee verzamelingen:

$$\begin{aligned}\mathcal{S}_i &= \mathcal{U}(G_i) \\ \mathcal{P}_i &= \{m(\mu) \in \mathcal{S} \mid \exists G_j : j < i, \mu \in \mathcal{U}(G_j)\}\end{aligned}$$

Roep nu voor elke G_i de frequent set intersection routine op met parameters σ , z , \mathcal{S}_i en \mathcal{P}_i . De verzameling \mathcal{P}_i van hierboven vormen de constraints bij het frequent set intersection algoritme. Het is een optimalisatie om de doorsnede van \mathcal{S}_i en \mathcal{P}_i niet in \mathcal{S}_i te voegen omdat deze verzamelingen door het algoritme toch niet zullen gebruikt worden om de doorsnede te berekenen.

We zouden willen dat de frequente doorsneden die worden geoutput kunnen beschouwd worden als frequente subgrafen. We moeten wel een aantal aanpassingen maken aan ons basisalgoritme van de frequent intersection miner omdat de verzamelingen nu matching-uiteinden zijn en dus een extra semantiek met zich meedragen. Zij \mathcal{F} de huidige filter tijdens een recursieve oproep van het frequent set intersection mining algoritme. We definiëren de *graafvertegenwoordiging* in de filter als:

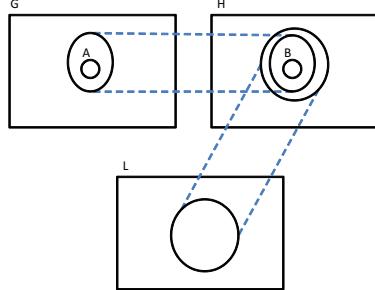
$$\mathcal{G}(\mathcal{F}) = \bigcup_{\mu \in \mathcal{F}} \{H \in \mathcal{G} \mid \mu \in \mathcal{U}(H, G_i)\}$$

De support van de filter definiëren we als $|\mathcal{G}(\mathcal{F})| + 1$. We doen +1 omdat G_i niet expliciet vertegenwoordigd is in de filter, maar wel steeds aanwezig is.

Wanneer we twee match-uiteinden toevoegen die eenzelfde graaf vertegenwoordigen, neemt de support niet toe. Het heeft immers weinig zin om twee matching-uiteinden van eenzelfde graaf toe te voegen aan de filter omdat dit over de graaf-populatie bekeken de resulterende doorsnede niet vaker doet voorkomen in verschillende grafen. Bovendien zou dit de doorsnede dan ook onnodig doen krimpen. We breiden de filter enkel uit met verzamelingen die de support strikt doen toenemen.

3.2.3 Bespreking

Beschouw een frequente doorsnede $A \subseteq V(G_i)$. Beschouw de matching-uiteinden $\mu \in \mathcal{U}(G_i)$ waarvoor $A \subseteq \mu$. Zij $r : V(G_i) \rightarrow H$ de matching die gedefinieerd



Figuur 3.1: Drie grafen G, H, L en een intersectie-patroon A gevonden in G .

is op μ . Het beeld van A onder de matching, B , is een deelgraaf in H . Zij σ' de support van doorsnede A in G_i . Dan weten we dat er σ' aantal van zulke H bestaan. Voor elk paar knopen $a_i, a_j \in A$ worden door de matching de bogen bewaard in B en omgekeerd. Bijgevolg is A een frequent induced subgraph.

Beschouw Figuur 3.1. Zij A het intersectie-patroon dat is gevonden in G . Laten we even veronderstellen dat A geconnecteerd is. Zij B het isomorfe overeenkomende patroon in een andere graaf H , door een matching tussen G en H . Is het mogelijk dat als we in H hadden gezocht naar frequente doorsneden, dat we ook het patroon B zouden gevonden hebben, maar dat de support van B in H groter is dan de support van A in G ? Stel even dat dit mogelijk zou zijn. Dit betekent dat er een andere graaf L bestaat zodat B in een matching tussen H en L geprojecteerd wordt naar een deelgraaf van $C \subseteq L$, maar dat L geen matching heeft van een deelgraaf van zichzelf naar een deelgraaf $A' \subseteq G$ zodat $A \subseteq A'$. De matching die B naar C afbeeldt is mogelijk gedefinieerd op nog meer knopen, maar dat maakt op zich niet uit. We weten dat A isomorf is met B en dat B isomorf is met C . Hierdoor is A isomorf met C en zou er toch een matching bestaan tussen G en L . Bijgevolg kan er zo geen L bestaan. Het maakt dus niet uit in welke graaf een frequente doorsnede wordt gevonden, omdat dit de support niet beïnvloedt van de bijhorende frequent subgraph.

Het is wel mogelijk dat door het zoeken van frequente doorsneden in H we een patroon B zouden kunnen vinden dat de projectie van A strikt omvat. Maar in dat geval is de support van B lager dan de support van A . Immers, stel dat de support groter of gelijk zou zijn aan die van A , dan kunnen we op een gelijkaardige manier als hierboven argumenteren dat er dan ook isomorfismen (=matchings) zouden moeten bestaan van andere grafen naar A waardoor zijn support altijd minstens zo groot is als die van B .

Het is mogelijk dat A niet geconnecteerd is, maar neem in bovenstaande redenering dan als A telkens een geconnecteerd deelstuk, dat aan de minsize threshold voldoet.

Stelling: bij het zoeken naar frequent intersections in de graaf G vindt men alle frequente subgraphs die in G bevatten zijn. Stel even dat dit niet zo zou zijn. Stel dat we in een graaf H een intersectie-patroon B zouden vinden dat weliswaar bevat is in een matching-uiteinde van G , maar dat dit patroon niet isomorf is aan een patroon dat we vinden in G zelf. Veronderstel voor de eenvoud even dat B geconnecteerd is. Beschouw alle matchings vanuit H naar andere grafen L waarbij B bevat is in een matching-uiteinde. Noem de pro-

jecties van B in deze andere grafen C . De projectie van B in G noemen we zoals voorheen A . De isomorfie tussen A en deze C 's duidt op een (mogelijk grotere) paarsgewijze matching tussen de grafen G en L . De overlap van al deze matching-uiteinden in G bevat het patroon A en is frequent als B frequent is. Dit patroon gaan we zeker ook vinden in G als we tenminste alle mogelijk paarsgewijze matchings zouden gevonden hebben van G met alle andere grafen. Bovenstaande redenering kan men ook maken indien B niet geconnecteerd is, namelijk op elk afzonderlijk geconnecteerd deel.

De verzameling \mathcal{P}_i van hierboven vormen de constraints. Beschouw een graaf G_j met $j < i$ en zij $\mu \subseteq V(G_j)$ een matching-uiteinde. Omdat $m(\mu)$ isomorf is aan μ mogen we in G_i geen patronen meer vinden die een deelverzameling vormen van $m(\mu)$ omdat we dan deze patronen ook al zouden gevonden hebben in G_j . Stel dat we deze constraints zouden weglaten en een patroon A zouden vinden in G_i dat bevat is in een constraint van een verzameling G_j en niet isomorf is aan een patroon gevonden in G_j . We kunnen via de omvattende matching-uiteinden A projecteren naar alle grafen die dit patroon bevatten, inclusief G_j omdat A bevat is in een constraint (=matching-uiteinde) van G_j . We hebben hierboven geargumenteerd dat door het zoeken naar patronen in G_j we reeds alle patronen hadden gevonden waarin G_j meedoet, dus A kan niet nieuw zijn.

Opmerkingen. Voor elke G_i kunnen we de frequent set intersections van de matching-uiteinden in principe in parallel berekenen.

3.2.4 Het patroon kan splitsen

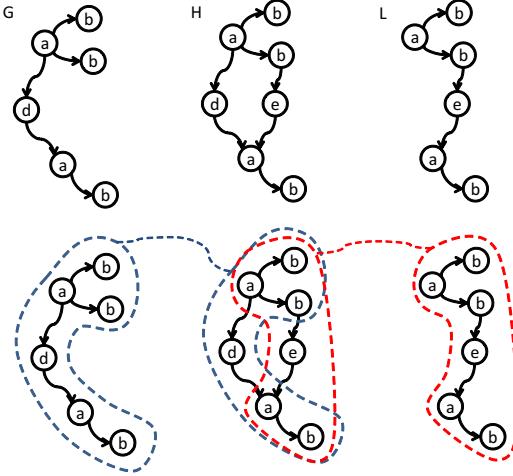
We zijn voornamelijk geïnteresseerd in frequent subgraphs die geconnecteerd zijn, omdat deze in het algemeen duidelijker de verbanden tussen de betrokken knopen aangeven en daarom eenvoudiger begrijpbaar zijn en nuttiger voor een potentiële toepassing.

Het is door het zoeken naar frequente doorsneden in een graaf G_i echter mogelijk dat we patronen bekomen die niet geconnecteerd zijn. Beschouw bijvoorbeeld de drie grafen in Figuur 3.2.

Het zou in het algemeen vermoedelijk overmatig restrictief zijn om bij het minen naar frequente doorsneden enkel uitbreidingen μ te doen aan de filter \mathcal{F} wanneer de projectie van μ binnen $\mathcal{F}(V(G_i))$ geconnecteerd is.

Het is echter wel niet duidelijk wat er moet gebeuren met een niet-geconnecteerd patroon. We zouden misschien tijdens het berekenen kunnen merken bij welke filter-uitbreiding het patroon splitst en dan recursief kunnen verderwerken op de geconnecteerde delen afzonderlijk (indien die groot genoeg zijn). Stel dat we in Figuur 3.2 zouden beginnen met de minen van doorsneden in H . Dan vinden we tijdens het minen twee geconnecteerde frequente patronen van gelijke support: $a - b - b$ en $a - b$. Hierbij is $a - b$ dan eigenlijk redundant. We zouden bijvoorbeeld wel het patroon $a - b - b$ willen vinden, en dan niet $a - b$ afzonderlijk. Het is niet meteen duidelijk hoe we dit kunnen doen. Het probleem doet zich eigenlijk voor omdat het geheel van $a - b - b$ en $a - b$ *tesamen* een frequent niet-geconnecteerd patroon is, dat we misschien niet mogen ontbinden in verbonden deel-patronen?

Het is door het splitsen van patronen ook mogelijk dat sommige patronen meermaals worden gemeld. Een niet-geconnecteerd patroon A bestaat dan uit afzonderlijke geconnecteerde componenten B , en het uniek zijn van A hangt dan af van het uniek samen voorkomen van deze geconnecteerde componenten.



Figuur 3.2: Drie grafen, waarbij het frequente intersectie-patroon in H niet geconnecteerd is.

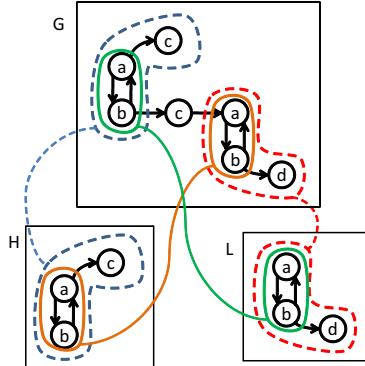
Het is echter niet uit te sluiten dat dezelfde componenten B kunnen ontstaan op verschillende plaatsen door het intersecteren van matching-uiteinden die structureel weinig met elkaar te maken hebben. Omdat we enkel matchings hebben beschouwd waarbij de matching-uiteinden geconnecteerd zijn hebben we via de constraint-verzameling \mathcal{P}_i geen controle op vermijden van dubbele niet-geconnecteerde patronen. Een mogelijke oplossing is eventueel het beschouwen van ook matchings zonder deze beperking, maar dan neemt de hoeveelheid mogelijke matchings sterk toe.

3.2.5 Omgaan met subgraaf automorfismen

Indien er subgraaf automorfismen mogen voorkomen in de grafen is het mogelijk dat er frequente subgraphs worden gemeld die isomorf aan elkaar zijn. Dit wordt geïllustreerd in Figuur 3.3. De graaf G bevat een subgraaf automorfisme op het knopen-paar gelabeld met a en b . Dit is niet apart getekend. In het blauw en oranje zijn de twee matchings tussen G en H gevisualiseerd. In het groen en rood zijn de twee matchings tussen G en L gevisualiseerd. Stel dat we de minsup gelijkstellen aan 2 en dat we in G zoeken naar de frequente doorsneden. We vinden de doorsnede van de groene en blauwe matching als een frequente doorsnede en ook de doorsnede van de oranje en rode matching. Beide patronen zijn echter isomorf. Het zou geen ramp zijn om deze allebei te vinden, omdat via post-processing men de dubbels er kan uitvissen. We kunnen echter proberen om dit te vermijden. Daarvoor beschrijven we in deze paragraaf een mogelijk voorstel.

3.2.5.1 Actieve matching-uiteinden

Bereken voor elke graaf G de zelf-matchings. We beschouwen enkel zelf-matchings waarbij de matching-uiteinden op minstens z knopen gedefinieerd zijn en waarbij de matching-uiteinden geconnecteerd zijn. We gaan sommige van de bijhorende



Figuur 3.3: grafen G,H,L waarbij G een subgraaf automorfisme bevat.

matching-uiteinden als constraints meegeven aan de frequent set intersection routine.

Zij \mathcal{U}_s de verzameling van alle matching-uiteinden μ van de zelf-matchings in G . We gaan deze eerst groeperen. Voor elke μ definiëren we zijn *groep* als de verzameling $R(\mu) = \{\mu' \in \mathcal{U}_s \mid \mu = \mu' \vee \mu = m(\mu')\}$. Wanneer twee μ 's in \mathcal{U}_s isomorf zijn, zitten ze in dezelfde groep. De doorsnede van twee groepen is leeg.

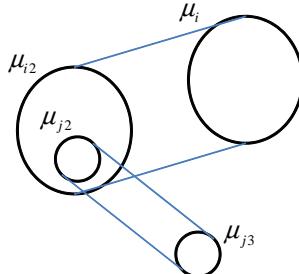
Verzamel al zulke unieke groepen in de groepen-verzameling \mathcal{R} . Zij R_1, R_2 twee unieke groepen uit \mathcal{R} . We definiëren dat groep R_1 de groep R_2 *bevat* als $\exists \mu \in R_2, \mu' \in R_1 : \mu \subsetneq \mu'$. Het is niet mogelijk dat $\mu = \mu'$ omdat μ dan in R_1 zou moeten zitten en de doorsnede tussen twee verschillende groepen is leeg. We noteren $R_2 \subset R_1$.

Beschouw nu alle groepen R_i die niet bevatten zijn in een andere groep. Dit noemen we de top-level groepen. Kies een willekeurig groepslid $\mu_i \in R_i$ en voeg dit toe aan een verzameling \mathcal{A} van *actieve matching-uiteinden*.

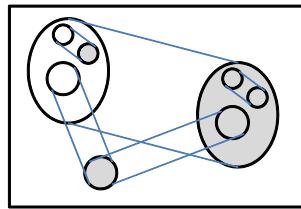
Beschouw dan alle $R_j \subset R_i$ die niet bevatten zijn in een andere $R_j \subset R_i$, m.a.w. de top-level groepen binnen R_i . Dit noemen we de rechtstreekse kinderen van R_i . Er bestaat nu een μ_j zodat $\mu_j \subset \mu_i$. Stel dat dit niet zo zou zijn. Beschouw de schets in Figuur 3.4. Er moeten sowieso een $\mu_{j2} \in R_j$ en een $\mu_{i2} \in R_i$ bestaan zodat $\mu_{i2} \neq \mu_i$ en $\mu_{j2} \subset \mu_{i2}$, want anders zou niet gelden $R_j \subset R_i$. Bovendien bestaat er nog een $\mu_{j3} \in R_j \setminus \{\mu_{j2}\}$ waarvoor niet geldt $\mu_{j3} \subset \mu_{i2}$ of $\mu_{j3} \subset \mu_i$ omdat anders de matching tussen μ_{i2} en μ_i de matching tussen μ_{j2} en μ_{j3} overbodig zou maken en dus het bestaan van μ_{j2} teniet zou doen. Er is een zelf-matching tussen μ_{j2} en μ_{j3} . We weten dat door de matching tussen μ_{i2} en μ_i de μ_{j2} eigenlijk ook een beeld heeft in μ_i . Dit beeld is de gezochte μ_j en dit matching-uiteinde bestaat effectief als afzonderlijke (deel)verzameling omdat de matching tussen μ_{j3} en μ_j bestaat.

We zijn daarom in staat voor alle toplevel groepen $R_j \subset R_i$ precies één matching-uiteinde $\mu_j \in R_j$ te selecteren dat bevat is in een actief matching-uiteinde geselecteerd voor de groep R_i . Voeg deze μ_j toe aan de verzameling van actieve matching-uiteinden \mathcal{A} . Herhaal dan de procedure recursief, door R_j in de plaats van R_i te nemen.

Soms is het echter mogelijk dat eenzelfde groep R_j van meerdere groepen het rechtstreekse kind is. In dat geval mag slechts één van deze parents bovenstaande procedure uitvoeren om een matching-uiteinde te selecteren voor R_j .



Figuur 3.4: Nesting van matching-uiteinde-groepen



Figuur 3.5: Dit is een abstract voorbeeldje van het (in)actief maken van de uiteinden van zelf-matchings. De grote rechthoek stelt de hele graaf voor en de ovalen stellen de uiteinden voor van zelf-matchings, die zelf in blauwe dunne lijnen zijn weergegeven.

Op het einde van deze procedure houden we een verzameling van actieve matching-uiteinden over.

3.2.5.2 Toepassing

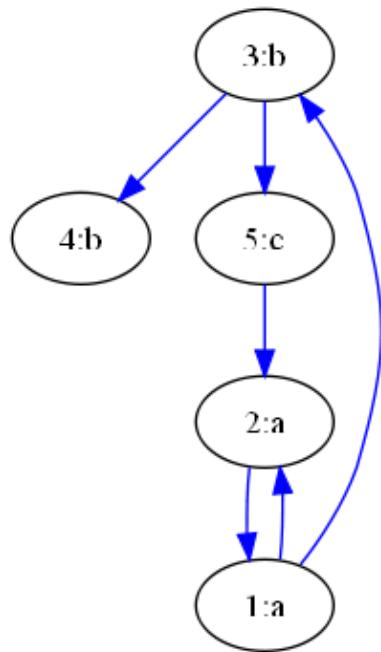
We keren nu terug naar het minen van frequent intersections in een graaf G_i . We breiden de verzameling constraints \mathcal{P}_i uit met alle matching-uiteinden van zelf-matchings die niet voorkomen in de verzameling \mathcal{A} . Deze matching-uiteinden zijn “niet actief”. In feite komt deze maatregel een beetje overeen met de oorspronkelijke inhoud van \mathcal{P}_i . Deze constraints drukken immers uit dat een verzameling knopen reeds “elders” is beschouwd en dat we daar dan niet meer naar moeten kijken.

In Figuur 3.5 is een abstract voorbeeld geschetst van het (in)actief maken van bepaalde matching-uiteinden van zelf-matchings.

3.3 Voorbeeld

Het basisalgoritme van het zoeken naar frequente subgrafen wanneer het patroon niet splitst hebben we al getest. We hebben nog geen maatregelen genomen tegen subgraaf automorfismen. Het is ook nog nodig hier rigoreuzer op te testen. We geven hier al een voorbeeld van de output geproduceerd door ons algoritme.

Beschouw de vier grafen in Figuren 3.6- 3.9. Elke knoop in de getoonde grafen heeft een label van de vorm $x : y$. Hierbij is x louter bedoeld om de knopen uniek aan te duiden terwijl y het concrete label is waarop twee knopen

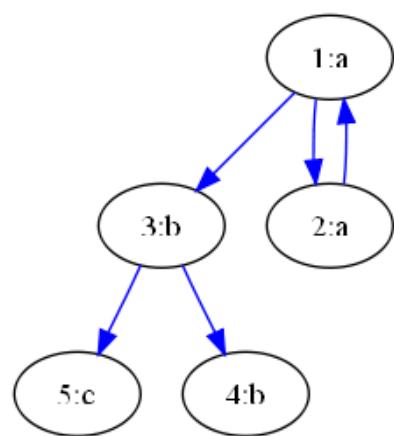


Figuur 3.6: Input-graaf g1

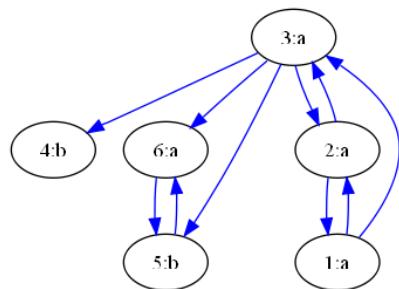
worden vergeleken bij het zoeken naar matchings. Als we $\sigma = 2$ en $z = 2$ nemen krijgen we via ons algoritme de frequent subgraphs getoond in Figuren 3.10 en 3.11. Bij elke frequent subgraph vermelden we naast zijn support ook telkens in welke input-graaf dit patroon werd gevonden via frequent set intersection mining op de matching-uiteinden. De frequent subgraphs hebben we daarom visueel weergegeven als een subset van de graph waarin zij zijn gevonden, waarbij we de knopen van de subgraph met groen gekleurd hebben.

De hier getoonde figuren werden via GraphViz¹ gegenereerd.

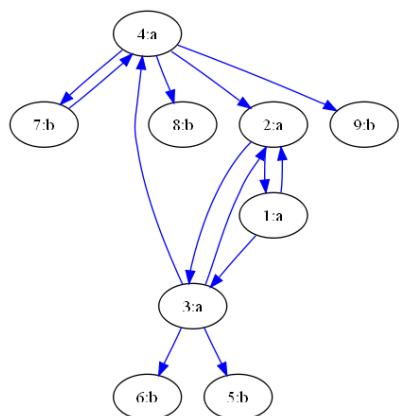
¹<http://www.graphviz.org/>



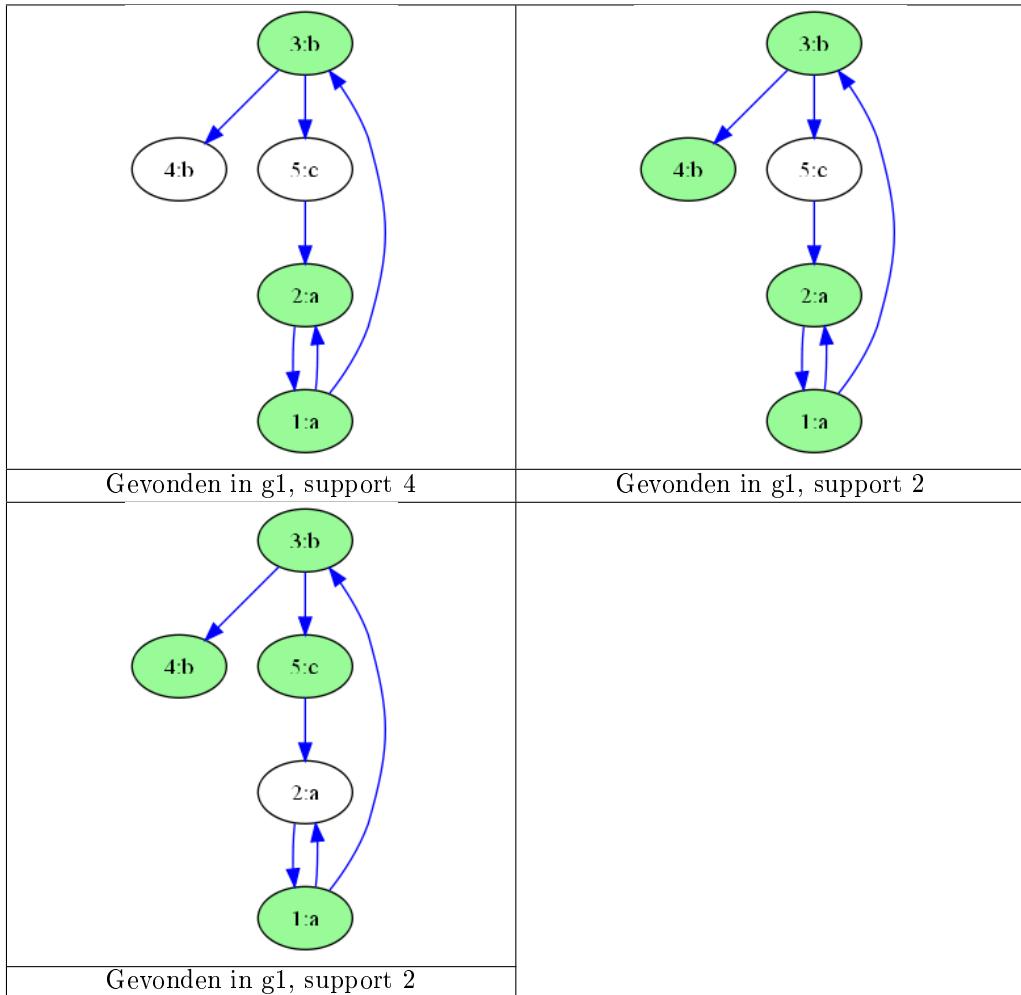
Figuur 3.7: Input-graaf g2



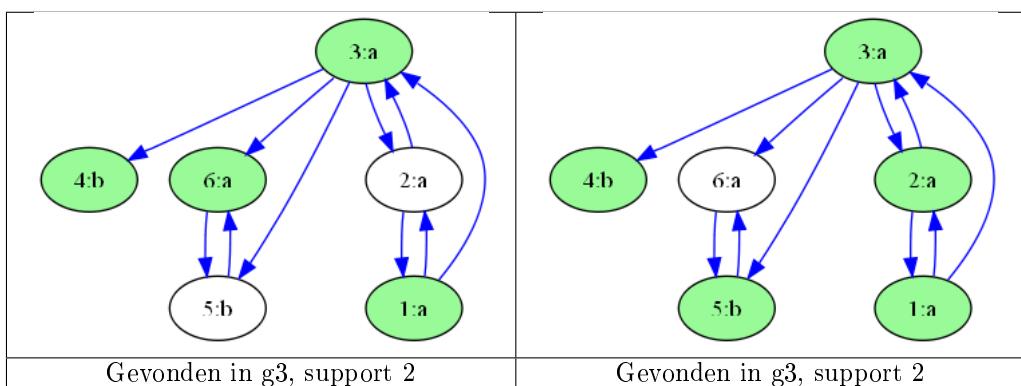
Figuur 3.8: Input-graaf g3



Figuur 3.9: Input-graaf g4



Figuur 3.10: Gevonden frequent subgraphs (deel 1)



Figuur 3.11: Gevonden frequent subgraphs (deel 2)

Hoofdstuk 4

Menselijke perceptie

In dit hoofdstuk bespreken we enkele concepten en modellen uit andere domeinen zoals psychologie en neurobiologie die we hebben leren kennen via een korte literatuur-studie. Dit heeft ons inspiratie gegeven voor het ontwikkelen van eigen ideeën, waaronder de ideeën die we reeds geschetst hebben in het inleidende hoofdstuk 1.

4.1 Visuele functionaliteit in de hersenen

In deze paragraaf geven we een ruwe schets van de werking van de hersenen op visuele prikkels die worden opgevangen in het oog. We volgen hiervoor het survey paper van Ng et al [9]. We citeren niet de concrete werken die erin vermeld staan, maar citeren gewoon het werk [9] voor een resultaat. Tenzij anders aangegeven komen alle feiten uit het werk [9]. De figuren zijn zelf gemaakt.

Visuele prikkels ontstaan in het oog. Het netvlies bevat *photoreceptors* die reageren op licht en samen samplen de photoreceptors het gezichtsveld. Daarnaast zijn er *ganglion* cellen die de prikkels van de photoreceptors bundelen en doorsturen naar de hersenen via de oogzenuw. De photoreceptoren en ganglion cellen variëren in grootte en eigenschappen afhankelijk van waar zij zich bevinden tov het centrum van het netvlies. Sommige ganglion cellen hebben een hoge ruimtelijke resolutie maar reageren traag op visuele veranderingen en andere cellen hebben een lage ruimtelijke resolutie maar kunnen vlugger visuele veranderingen waarnemen over de tijd. Dit verschil in temporele resolutie is van minder belang voor ons werk.

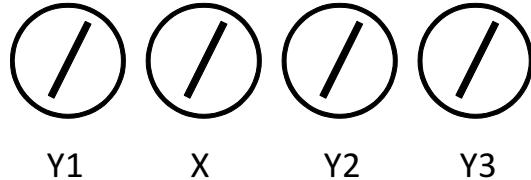
De *visual cortex* is een deel van onze hersenen en is verantwoordelijk van het doorgeven en verwerken van de visuele prikkels die ontstaan in het oog. De visual cortex is georganiseerd in anatomisch gescheiden *processing areas* [14, 10, 9]. Deze gebieden krijgen elk een aparte naam: V1, V2, V3, V4, V5/MT, MST, enz. Onderzoek heeft uitgewezen dat de visuele signalen vanuit het netvlies de visual cortex binnenkomen, vooral langs het gebied aangeduid als V1, ook wel *primary visual cortex* genoemd. In V1 worden de prikkels verwerkt en nieuwe prikkels die daar ontstaan reizen door naar andere gebieden van de visual cortex. V1 speelt een belangrijke rol in de menselijke perceptie. Er bestaan twee parallele *processing pathways* in de visual cortex, aangeduid als *dorsal pathway* en *ventral pathway*. De dorsal pathway volgt het spoor $V1 \rightarrow V2 \rightarrow V5 \rightarrow \dots$ en wordt

ook wel de *where pathway* genoemd omdat zijn functionaliteit gebruikt wordt in het localiseren van objecten, wat nodig is om ruimtelijke taken te kunnen uitvoeren. De *ventral pathway* volgt het spoor $V1 \rightarrow V2 \rightarrow V3 \rightarrow V4 \rightarrow \dots$, en wordt de *what pathway* genoemd. Deze zorgt voor het herkennen van objecten. Door de schijnbare volgorde in elke pathway, worden de processing areas die volgen op $V1$ ook wel aangeduid als “hoger gelegen” of “later”.

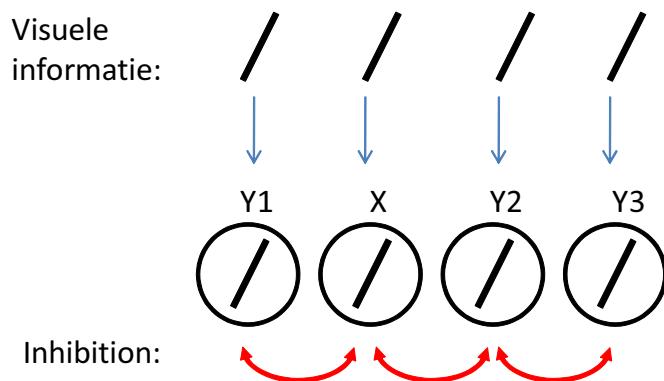
Elke zenuwcel in de visual cortex “reageert” op een klein deel van het gezichtsveld. Dit deel wordt het *receptive field* (RF) van de cel genoemd. De grootte van de receptive field van een cel neemt toe als we de where of what pathway zouden aflopen. Dit betekent dat cellen in hogere processing areas van de visual cortex informatie integreren over steeds grotere gebieden van het gezichtsveld.

$V1$ beslaat het grootste deel van de visual cortex. Men schat de verhouding van het aantal $V1$ zenuwcellen tot het aantal ganglion output cellen als 100 : 1. Elk type photoreceptor, zijn plaats op het netvlies en zelfs in welk oog hij zich bevindt geeft aanleiding tot verschillende types van prikkels. De ganglion cellen voeren een soort compressie uit op de visuele prikkels om deze efficiënt te kunnen doorsturen door de oogzenuw, en de cellen in $V1$ moeten deze informatie terug “uitpakken”. Maar daarnaast produceert $V1$ toch nog veel extra andere output, meer dan er visuele input is. $V1$ vormt een gedetailleerde en overcomplete representatie van het gezichtsveld door zijn reusachtige populatie van geprikkelde zenuwcellen. $V1$ bevat immers celtypen met verschillende soorten receptive fields en visuele eigenschappen. Er zijn types zenuwcellen die gevoelig zijn voor bepaalde *feature types*: oriëntaties van randen, verschuiving, schaal, kleur, ... De activatie van elk type zenuwcel zou men kunnen modelleren in functie van een lineaire combinatie van de prikkels die zich aandienen in zijn receptive field. De $V1$ zenuwcellen kunnen ook gevoelig zijn voor meerdere feature types tegelijk. Dit zou veroorzaakt kunnen worden door het feit dat de zenuwcellen gewoon een lineaire combinatie nemen van hun visuele inputs. Toch blijven de $V1$ zenuwcellen eerder getuned voor één specifieke feature type. Men vermoed dat de cellen en hun receptive fields in $V1$ kunnen begrepen worden als een efficiënte codering die het visuele signaal “uit elkaar rafelt”. $V1$ zou door de evolutie zodanig aangepast zijn dat het menselijk brein heel efficiënt visuele features kan detecteren en (intern) voorstellen.

De zenuwcellen die gevoelig zijn voor oriëntatie kunnen we kort belichten. Zogenaamde *simpel* zenuwcellen worden geactiveerd wanneer lijnen of randen die georiënteerd zijn in een bepaalde richting opduiken in hun receptive fields. Deze “richting” wordt uitgedrukt in een denkbeeldig lokaal (x,y)-assenstelsel dat bovenop het gezichtsveld ligt. De activatie-sterkte van een simpele cel hangt af van de verschuiving van de prikkel in het receptive field van zo’n cel. De receptive fields van deze cellen overlappen vrij sterk. Hieruit blijkt dat niet elke cel een gescheiden stukje van het gezichtsveld als input krijgt. Complexe cellen daar tegen zijn ook gevoelig voor een bepaalde oriëntatie van een lijn/rand, maar minder gevoelig voor de precieze verschuiving ervan binnen hun (iets grotere) receptive field. Dit zou bewerkstelligd worden door het feit dat de activatie van elke complexe cel een samenbundeling is van de activaties van een aantal simpele cellen in zijn receptive field. Elk van die simpele cellen is lichtjes anders geplaatst in de receptive field van de complexe cel. Voor een praktische implementatie van de concepten van simpele cellen en complexe cellen zie bijvoorbeeld [8, 15].



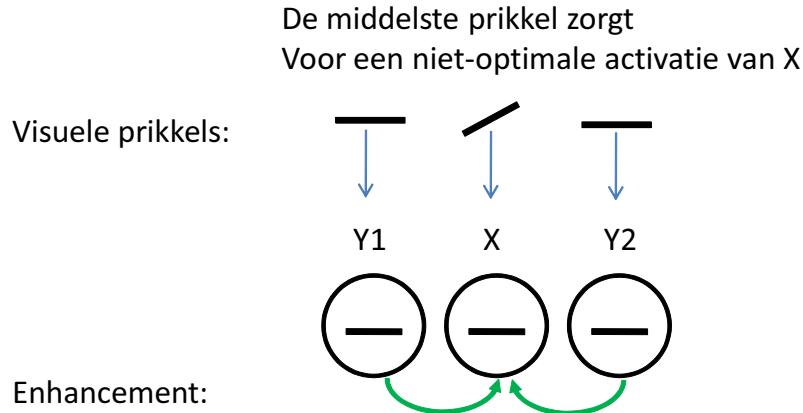
Figuur 4.1: Zenuwcellen, allen getuned op een rand/lijn in een bepaalde oriëntatie



Figuur 4.2: Illustratie van inhibition

Traditioneel bekijkt men de processing die plaatsvindt in de visual cortex als een denkbeeldige piramide waarbij informatie wordt geïntegreerd over steeds grotere deelgebieden van het gezichtsveld, vertrekende aan de basis (V1) en gaande tot de top (V4, V5). Dit komt omdat de receptive fields van de cellen alsmaar toenemen bij het afwerken van een processing pathway. Bovenaan de denkbeeldige piramide zijn dus minder cellen nodig om iets te zeggen over de hele visuele input. Elke zenuwcel in V1 heeft echter ook veel connecties naar andere zenuwcellen in V1, en deze connecties kunnen ook zorgen voor wederzijdse interacties in de eerste stadia van deze piramide verwerking. Er is dus niet alleen interactie tussen de anatomische gebieden van de visual cortex maar ook erbinnen. Elke zenuwcel wordt buiten de rechtstreekse prikkels in zijn receptive field ook in belangrijke mate beïnvloed door de activaties van andere cellen waarvan de receptive fields "in de buurt" liggen. Het gebied waaruit deze beïnvloeding afkomstig is noemt men de *context* of *surround* van een zenuwcel. Door de context is het mogelijk dat de activatie van een zenuwcel meer globale eigenschappen van de visuele input (van bijvoorbeeld een visueel voorwerp) kan weerspiegelen, eigenschappen die optreden in een veel groter gebied dan zijn eigen receptive field. Cellen die hetzelfde feature type herkennen en die zich in elkaar beïnvloedingsfeer bevinden gaan elkaar activatie onderdrukken (*inhibition*) of juist versterken (*enhancement of facilitation*).

Beschouw in het bijzonder een zenuwcel X die getuned is op het herkennen van een rand/lijn die georiënteerd is volgens richting O . In de surround van X zitten andere zenuwcellen Y_i die getuned zijn op precies datzelfde soort feature type. Veronderstel dat de schikking van de cellen is zoals in Figuur 4.1. De



Figuur 4.3: Illustratie van enhancement

ligging van de cellen in de figuur komt overeen met de ligging van hun receptive fields in het gezichtsveld. Stel dat in de receptive fields van cellen X en Y_i zich een optimale prikkel aanbiedt. Een optimale prikkel geeft aanleiding tot maximale activatie van een cel. Men weet dat de activatie van X wordt onderdrukt door de naburige cellen Y_i , omdat de cellen Y_i ook maximaal geprikkeld worden, dit heet *isoorientation inhibition*. Dit is getoond in Figuur 4.2. In deze figuur hebben we voor de eenvoud verondersteld dat enkel cellen die naast elkaar getekend zijn in elkaars beïnvloedingsgebied liggen. Merk op dat de cellen Y_i elkaar activatie dus ook belemmeren. Dit effect is zwakker tussen cellen die verschillende oriëntaties herkennen en het zwakste tussen cellen die getuned zijn op onderling loodrechte richtingen.

Zoals hierboven aangegeven kunnen cellen elkaar activatie echter ook versterken. Opnieuw beschouwen we het voorbeeld van oriëntatie-getunedede cellen. Er is een nieuwe situatie getoond in Figuur 4.3. Hier is cel X niet optimaal geactiveerd omdat de visuele prikkel niet perfect overeenkomt met de de oriëntatie waarop X getuned is. Maar cellen Y_1 en Y_2 worden wel sterk geprikkeld. Merk op dat de feature types waarop X , Y_1 en Y_2 getuned zijn collineaire randen/lijnen helpen voorstellen. In dat geval gaan zij elkaar versterken. Het resultaat is dat de initiële lage activatie van X op zijn receptive field vervangen wordt door een sterker activatie als resultaat van facilitation uit zijn surround. Tesamen stellen de activaties van deze drie cellen nu misschien een lichtjes onzuivere rechte lijn voor of (als een deel van een contour). Dit is een mooi voorbeeld hoe lokale informatie in elkaar gepuzzeld wordt om grotere features af te leiden zoals omtrekken en oppervlakken. De connecties die plaatsvinden tussen zulke cellen in V1 over langere afstanden dienen meestal ter facilitation terwijl connecties over kortere afstanden meestal voor inhibition dienen. De facilitation gebeurt niet alleen tussen cellen die samen een rechte lijn herkennen, maar ook tussen cellen die zachtjes een curve vormen [3].

Het is mogelijk dat V1 de mechanismen van inhibition en facilitation nodig heeft om belangrijke “verschillen” op te merken. Dit wordt gemodelleerd met een *saliency map*, die grotendeels gebaseerd is op het contrast van prikkels. Alle effecten van isoorientation inhibition, collineaire facilitation, kleurcontrasten,

... worden hierin samengebracht. Enkel de cellen met een sterke activatie dragen hiertoe bij. De saliency map zou het resultaat kunnen zijn van voor elke plaats in het gezichtsveld een soort maximum of som te nemen over de activaties van de cellen die daar hun receptive field hebben. Dit model stelt voor dat V1 al een soort rudimentaire segmentatie uitvoert op de visuele input om interessante visuele verschillen op te merken. Deze verschillen kunnen onbewust de aandacht van het oog trekken en de rest van de visual cortex. Daarom noemt deze segmentatie (binnen de psychologie) *preattentive segmentation/selection*. Zhaoping en Dayan schreven hierover een klein maar interessant artikel met mooie voorbeelden [18].

4.2 Andere noemenswaardigheden

Kietzmann, Lange en Riedmiller spreken over een belangrijk begrip, namelijk *Hebbian learning* [5]. Hebbian learning stelt dat wanneer twee prikkels vaak samen voorkomen, hun herinneringen in het geheugen sterk aan elkaar zullen gelinkt zijn. In het artikel [5] wordt een toepassing besproken waarbij verschillende visuele standpunten van eenzelfde object aan elkaar kunnen gekoppeld worden door Hebbian learning in het visuele geheugen. Dit is bij mensen typisch het geval omdat wij een associatief geheugen hebben. Associaties kunnen niet alleen visueel zijn, maar ook in de tijd. In [5] wordt voorgesteld en gemootiveerd dat artificiële systemen bij het waarnemen van een ruimtelijk voorwerp geen interne 3D-voorstelling moeten maken van dit voorwerp, maar gewoon de verschillende 2D-beelden met elkaar kunnen associëren.

Men kan objecten visueel in de diepte roteren of in de image plane. Bij de eerste soort rotatie verandert de kijkhoek op het object. Bij de tweede soort rotatie draaien we het beeld rond de kijk-as van het gezichtsveld. Het herkennen van objecten nadat zij in de diepte geroteerd zijn noemt men *view-invariant object recognition*. In het artikel [2] wordt experimenteel bewijs aangeleverd voor het feit dat mensen omgaan met visuele prikkels van objecten op een manier die neigt naar *view-dependent*. Dit wil zeggen dat de cellen in de visual cortex slechts in bepaalde mate visuele prikkels kunnen vergelijken met herinneringen nadat een view-verandering is opgetreden. De vergelijking die cellen uitvoeren op visuele prikkels tussen twee inputs van het gezichtsveld en het vinden van mogelijke overeenkomsten noemt men in [2] *cross-adaptation*. Deze informatie kan ons motiveren om ons te concentreren op het maken van view-afhankelijke visuele herinneringen voor ons systeem ipv view-onafhankelijke herinneringen.

Ook in de context van Hebbian learning, wordt in een ander artikel [16] bewijs gegeven voor het feit dat mensen visuele prikkels over schijnbaar dezelfde entiteit aan elkaar linken als deze na elkaar worden getoond. Dit is bijvoorbeeld het geval in een animatie waarin een gezicht in de diepte roteert op een scherm. Deze gezichten zijn afkomstig van 3D scans van echte personen. Men concludeert in dit artikel dat wanneer het ene gezicht zeer traag wordt *gemorpt* naar een ander gezicht dat de proefpersoon toch denkt dat dit bij één persoon hoort.

In het artikel [4] worden de resultaten besproken van een test waarbij mensen snel moesten aangeven of er in een afbeelding een dier aanwezig was of niet. De afbeeldingen werden gepresenteerd in vele rotaties (in het beeldvlak). Het artikel geeft aan dat er veel bewijs is in de literatuur dat mensen opvallend trager visuele voorwerpen herkennen wanneer zij geroteerd (in het beeldvlak)

gepresenteerd worden, maar dat voor andere taken dit schijnbaar niet waar is. Bij het experiment dat in [4] wordt besproken moesten immers geen concrete dieren herkend worden, maar moest gewoon de vraag beantwoord worden “is er een dier in deze afbeelding”. Hiervoor kan de mens vermoedelijk gebruik maken van meer algemene visuele technieken, zoals de verdelingen van oriëntaties. Ook is het eventueel mogelijk dat lokale features op basis van ervaring in geringe mate in elkaar gepuzzeld worden, zonder dat er iets globaals herkend wordt. Bijvoorbeeld: een vin, een poot, een stukje vacht zijn duidelijke aanwijzingen voor de aanwezigheid van een dier, zonder dat je duidelijk weet welk dier.

In het artikel [12] wordt bewijs aangedragen voor het feit dat mensen afhankelijk van de “taak” zeer snel bepaalde visuele objecten in een afbeelding kunnen terugvinden die slechts voor korte tijd wordt getoond. Het is zelfs mogelijk om deze objecten op te merken als er niet bewust naar gekeken wordt, maar als deze zich gewoon ergens in het gezichtsveld bevinden. Hiervoor zouden bepaalde gebieden in onze hersenen verantwoordelijk zijn die het verwerken van visuele informatie *biasen* op bepaalde categorieën van objecten. Door dit mechanisme worden de visuele prikkels gefilterd en enkel visuele informatie die hoort bij bepaalde objecten die relevant zijn voor een taak blijven over. Een voorbeeld (ook uit het artikel): bij het oversteken van een straat merken we vlug verkeer op waar we rekening mee moeten houden, zoals auto’s en fietsers. Op dat moment zijn we ons niet echt bewust van andere objecten in ons gezichtsveld, zoals huizen, gordijnen achter ramen, bloempotten, . . . De werking van dit mechanisme zou kunnen gebaseerd zijn op het volgende. Het bewustzijn activeert taak-specifieke object-categorieën uit het geheugen en deze categorieën biasen de cellen die visuele prikkels verwerken om zich enkel te concentreren op visuele eigenschappen die te maken hebben met de taak-specifieke object-categorieën. Deze informatie kan ons eventueel motiveren om ons eigen systeem te voorzien van een bias-mechanisme dat enkel objecten in foto’s zal proberen te herkennen die horen bij een bepaalde “hoofd” categorie die op voorhand door de gebruiker is geselecteerd. Bij het taggen van strandfoto’s kan de gebruiker bijvoorbeeld hoofdcategorie “vakantie” selecteren waardoor het systeem bijvoorbeeld visuele herinneringen aan astronauten en space shuttles zal uitsluiten en het taggen eventueel sneller kan verlopen.

In het artikel [5] worden ook een aantal interessante onderwerpen omtrent visual object recognition gebundeld om een praktische implementatie te maken waarmee robots in principe hun omgeving zouden kunnen begrijpen. Typisch aan de aanpak die daar wordt beschreven is dat men initieel een vast (groot) aantal feature types beschrijft en per database van afbeeldingen gaat *lernen* welke feature types voldoende zijn om de categorieën die daarin voorkomen te differentiëren. Wanneer bijvoorbeeld alle visuele voorwerpen in een database geel gekleurd zijn, dan zal het feature type dat de kleur beschrijft niet worden geselecteerd door het learning proces maar zullen feature types nuttig blijken die de vorm beschrijven. In het andere extreme geval, wanneer alle visuele voorwerpen in de database dezelfde contour/vorm hebben, zal het feature type “kleur” eerder belangrijk zijn om de categorieën te onderscheiden dan feature types die zich bezig houden met de vorm. Bij ons eigen systeem speelt dit feature type selectie proces eigenlijk niet mee. We moeten in principe kunnen omgaan met alle mogelijke afbeeldingen en dus moeten we in principe zoveel mogelijk visuele feature types toelaten om de visuele objecten mee te beschrijven.

Mensen zijn in staat om bij het waarnemen van een afbeelding omtrent een

scene vlug te concluderen of deze scene zich afspeelt in een omsloten geheel (zoals een kamer, een gang), of dat deze scene zich afspeelt in de “open lucht” en allerlei tussenvormen. Het artikel [10] introduceert een computationeel model dat dit menselijke vermogen moet nabootsen. Men introduceert een soort “globale” features die worden afgeleid via learning op een (grote) verzameling afbeeldingen. Dit gebeurt concreet door via *principal components analysis* een basis af te leiden voor een vectorruimte die gebaseerd op deze verzameling afbeeldingen. Deze basis bevat zowel lage als hoge resolutie componenten. Nieuwe afbeeldingen kunnen vervolgens begrepen worden door projectie op deze basis. De auteurs van [10] nemen een interessant standpunt in. Zij argumenteren dat een groot deel van de menselijke herkenning kan gebaseerd zijn op features die worden afgeleid uit een afbeelding *zonder* dat er eerst een segmentatie wordt uitgevoerd.

Verdergaand om het vorige artikel, hebben we nog volgende opmerking. In artikels [14, 11] wordt gesproken over het “kip-of-ei” probleem: wat gebeurt er eerst, low-level begrip van een afbeelding of high-level begrip van een afbeelding? Met low-level begrip wordt bedoeld dat de afbeelding een beetje gesegmenteerd wordt in lokale features die worden gecombineerd tot grotere gehelen die uiteindelijk op hun beurt (hopelijk) voldoende overeenkomen met visuele herinneringen. Met high-level begrip wordt bedoeld dat lokale features worden “overgeslagen” en meteen een soort globale prikkels een bepaalde verzameling visuele herinneringen selecteren waarna meer lokale features kunnen gebruikt worden voor uiteindelijke en meer zekere herkenning. Low-level begrip is lastig omdat lokale features ambigu zijn en meer globaal begrip nodig hebben om een duidelijke lijn te trekken. High-level begrip is schijnbaar ook pas mogelijk als er voldoende low-level informatie beschikbaar is, want anders is er “niets” om mee te beginnen. Voor ons eigen werk is dit kip-of-ei probleem uitermate belangrijk. De meeste artikels die werden bekeken voor dit tweede deel van de thesis beginnen echter met low-level begrip, waarschijnlijk omdat deze manier het meest intuïtief is. We denken deze aanpak te volgen.

4.3 Visuele informatie beschrijven

Verdergaand op de kennis uit vorige paragraaf bespreken we hier onze eigen modelvorming en opvattingen omtrent het beschrijven van visuele informatie.

4.3.1 Feature types

Het aantal photoreceptoren in het menselijk oog is eindig en min of meer constant. Daarom beperken we de input van ons programma tot een eindige verzameling van *receivers*. Deze receivers stellen het *gezichtsveld* voor van ons systeem. In toekomstige systemen kan de hoeveelheid receivers misschien wel heel groot zijn, wanneer er snelle hardware op de markt komt en/of camera’s met hoge resolutie. Opgelet: het menselijk gezichtsveld is in normale omstandigheden bijna altijd even groot. Daarom is de verzameling van receivers ook altijd even groot.

Hoe kunnen we de receivers eigenlijk voorstellen? We mogen ons systeem helemaal van scratch bouwen, dus we hebben in principe de vrijheid. In principe kunnen we ons systeem alle mogelijke inputs laten ontvangen die we kunnen meten met een of ander toestel, inclusief inputs die de mens zelf niet kan ontvangen. Onze toepassing is het werken met afbeeldingen en in dat geval lijkt het

echter voldoende om te werken met receivers voor kleur en intensiteiten. Onze receivers zouden dus gewoon pixels kunnen zijn in een grid. Het gezichtsveld is dan de kleinste rechthoek rond de pixels. Doordat een receiver een pixel is, heeft hij een bepaalde locatie (x, y) in het gezichtsveld, met $x, y \in \mathbb{N}$. Zoals voorheen in deze thesis is “linksboven” in het gezichtsveld de locatie $(0, 0)$. Laten we veronderstellen dat locaties (x, y) met $x, y \in \mathbb{R}^+ \setminus \mathbb{N}$ niet waarneembaar zijn voor het systeem. Wat gebeurt er als we een afbeelding aan ons systeem willen laten zien die te klein is voor het gezichtsveld? We zouden sommige receivers in dat geval geen input kunnen sturen en gewoon in *idle mode* kunnen laten. Wat gebeurt er als we daarentegen een afbeelding aan ons systeem willen laten zien die te groot is voor het gezichtsveld? Wij als mensen zouden bij het bekijken van een grote poster een paar stappen achteruit kunnen doen. Voor ons systeem zouden we een klein stukje van een te grote afbeelding kunnen laten zien of de afbeelding kleiner maken.

Eventueel kunnen we de verschillende pixel-componenten scheiden in verschillende layers, zoals bijvoorbeeld RGB of YUV.

Nu komt het er op neer om iets zinvolle te doen met de informatie die we ontvangen via de receivers. Volgens schoolse kennis zijn zintuigen bij organismen door de evolutie ontwikkeld om efficiënt de “omgeving” te kunnen analyseren en mogelijke gevaren en opportuniteiten op te merken. Om te kunnen reageren op zijn omgeving heeft een organisme ofwel instincten (een soort hardware programmatie) ofwel herinneringen (een soort software) waaruit het kan afleiden wat goed of slecht is. De noties “goed” en “slecht” drukken uit in welke mate aan bepaalde noden van het organisme is voldaan. Zonder voorgeprogrammeerde functionaliteit kan er in feite helemaal niets gebeuren. Een organisme zonder instincten kan niet leren, want besef van “goed” of “slecht” is hardware-functionaliteit (die je niet kan aanleren). Er is bijgevolg altijd wat nood aan een dun laagje hardcoded functionaliteit die al de rest moet *bootstrappen*. Laten we veronderstellen dat bij organismen het gezichtsvermogen aanwezig is als hardware functionaliteit. Al vrij vlug na de geboorte moet het gezichtsvermogen “zinvolle” informatie kunnen bekomen.

Volgens de evolutie-theorie van Darwin gebeuren doorheen de tijd in een organisme bepaalde veranderingen en de mate waarin het nieuwe organisme in zijn omgeving kan overleven vormt de “test” voor deze verandering. We doen geen uitspraak over het feit of hedendaagse organismen optimaal zijn aangepast aan hun omgeving of niet. Wat we wel weten is dat organismen rondom ons *werken omdat ze werken*. Terwijl deze tekst geschreven wordt is de werking van het menselijk zicht nog niet volledig ontrafeld, hoewel bepaalde componenten vermoedelijk worden begrepen (zie paragraaf 4.1). Bijgevolg moet ons eigen systeem volledig open staan voor toekomstige ontdekkingen uit het domein van de neurobiologie en dus eenvoudig uitbreidbaar zijn. Daarom definiëer ik een *feature* als alles wat men zou kunnen opmerken in het gezichtsveld. Dit kan natuurlijk creativiteit vergen van het systeem en daar zou men heel ver in kunnen gaan. Een feature is in deze zin dus een synoniem voor *eigenschap*. De beschrijving van een feature gebeurt door een feature type.

We mogen zelf bepalen hoe krachtig het systeem wordt en tegelijk moeten we het voor onszelf begrijpbaar houden zodat we het later nog kunnen verbeteren en uitbreiden. We moeten dus in tegenstelling tot de natuurlijke selectie van Darwin voor ons eigen systeem een soort *intelligent design* toepassen. Daarom kunnen we de features explicet modelleren en presenteren als (voor ons) be-

grijpbare entiteiten. Voorbeelden van feature types zouden dan kunnen zijn:

- een lijn
- een punt
- een veelhoek
- een lijn is krom/recht
- twee lijnen zijn parallel
- twee lijnen raken elkaar
- een feature is omsloten door een veelhoek
- een veelhoek is groter dan een andere veelhoek
- een veelhoek heeft een lichtere intensiteit dan een andere veelhoek
- de lengte van een lijn
- de afstand tussen twee punten
- ...

Om het overzicht te bewaren, kunnen we trachten een opdeling te maken. We doen een poging:

1. *atomair geometrisch*: deelverzamelingen van pixels in het vlak die we kunnen aanwijzen en benoemen met een label. Bijvoorbeeld: lijn, punt, veelhoek, We zullen de voorkomens van deze soort ook wel atomaire (geometrische) features noemen.
2. *unaire kenmerken van atomaire geometrische features*:
 - (a) *predicaten*: (kwalitatieve) uitspraken doen over eigenschappen van features. Bijvoorbeeld: een lijn is krom/recht.
 - (b) *attributen*: metingen op een feature, door bepaalde eigenschappen ervan te quantificeren, bijvoorbeeld: een veelhoek heeft een bepaalde oppervlakte (Eenheid = breedte of hoogte van één pixel).
3. *binaire relaties*: uitspraken doen over de binaire relatie tussen twee features. Bijvoorbeeld: veelhoek *A* heeft een grotere oppervlakte dan veelhoek *B* (vergelijken van attributen), of twee lijnen zijn parallel, of een feature zit omvat in een veelhoek, of de afstand tussen twee punten.
4. *N-aire relaties ($N > 2$)*: uitspraken doen over de relatie tussen meer dan twee features. Bijvoorbeeld: drie punten zijn collineair of vormen een driehoek, of de som van de oppervlakten van een aantal veelhoeken is kleiner dan 50.

De bovengenoemde relatie types plakken eigenlijk een label op een tuple (groep) van features.

Het is zeer natuurlijk om de atomaire geometrische features te beschouwen als “objectjes” of “entiteiten” want die kan je zien en “vastpakken”. Je kan de relaties daartussen ook beschouwen als aparte entiteiten, ook al zijn ze niet zo intuïtief zichtbaar. We hebben hier nog niets gezegd over de implementatie van de features, dus het is niet meteen duidelijk of we enkel de zichtbare features explicet zouden voorstellen in het computer-geheugen of ook de relaties daartussen.

Attributen van atomaire geometrische features hebben schijnbaar niet veel zin in isolatie. Je moet ze vergelijken om tot betekenis te komen. Sommige unaire relaties zou je kunnen vertalen naar een aantal binaire relaties. Bijvoorbeeld de uitspraak *deze lijn is krom* zou je kunnen vertalen naar de binaire relaties *de deel-segmentjes van de lijn zijn niet collineair*.

De N-aire relaties lijken al wat ingewikkelder. Deze spelen zich schijnbaar af na echte interpretatie en modelvorming over de input van het gezichtsveld. Een ander voorbeeld van feature type is: drie punten vormen een gelijkzijdige driehoek. In dat geval heb je dus de 3-aire relatie “niet-collinear” tussen de punten, maar ook het opmerken dat de paarsgewijze afstanden tussen deze punten gelijk zijn. Dit zou men kunnen modelleren met een 3-aire relatie “gelijk” over de drie afstanden (objectivering van de binaire relaties tussen de punten).

Belangrijke opmerking: we kunnen wel alle soorten informatie beschrijven met enkel binaire relaties, *als we nieuwe hulp-objecten mogen toevoegen*.

Het lijkt ook dat wanneer we in bovenstaande genummerde lijst van boven naar onder lezen, dat telkens de volgende soort feature type gebruik maakt van de feature types die daarvoor genoemd zijn. De binaire relaties vergelijken bijvoorbeeld attributen en attributen zijn gespecificeerd voor geometrische features. Worden de feature types van boven naar onder dus steeds complexer?

Misschien moeten we toch maar ergens een lijn trekken tussen features en niet-features. Features zouden we volledig bij de hardware functionaliteit van ons systeem kunnen rekenen en alles wat meer creativiteit/interpretatie vergt wordt een niet-feature. In dat opzicht, is het herkennen van een visuele gelijkbenige driehoek een feature of is dat iets wat je *geleerd* hebt? Komen gelijkbenige driehoeken voor in de natuur? Ja, maar niet zo vaak en eerder per toeval waarschijnlijk. Features zouden op zich niet veel interpretatie mogen vergen, want dat is de topic van een groter AI-systeem. Het herkennen van een gelijkbenige driehoek lijkt al iets meer begrip/intelligentie te vergen van de situatie of context waarin de visuele informatie zich bevindt. Ik stel daarom voor dat N-aire relaties met $N > 2$ eigenlijk een soort zachte creatieve “kennis” vormen van het systeem en dus geen features zijn. Daarom zijn de beschrijvende elementen die we voor ons systeem zullen beschouwen:

1. atomaire (geometrische) features
2. binaire relaties tussen deze atomaire geometrische features

We zullen binaire relaties niet meer beschouwen als features en “feature” reserveren voor de zichtbare entiteiten.

Elke feature type wordt gezocht met een specifiek algoritme en gebruiken mogelijk een soort geoptimaliseerde pipeline. Dit is geïnspireerd op de gespe-



Figuur 4.4: Streepjes en onzekere lijn

cialiseerde cellen en pipeline in de hersenfunctionaliteit zoals gezien in paragraaf 4.1. Voor de eenvoud kunnen we veronderstellen dat deze pipeline per feature type onveranderlijk is, onafhankelijk van beïnvloeding door keuzen van het systeem of omstandigheden. Soms kan het zoeken van het ene feature type vereisen dat er eerst voorkomens van andere feature types gevonden worden, bijvoorbeeld: het vinden van een feature “gesloten kromme” vereist dat je eerst features van type “rand” hebt gedetecteerd.

Voor de zekerheid maken we volgende opmerking. We veronderstellen dat features er zijn of er niet zijn, en niet “gedeeltelijk”. Als we een aantal streepjes achter elkaar zien staan, zoals in Figuur 4.4, dan denken we misschien wel: “Hé, deze streepjes vormen samen mogelijk een langere lijn”. Je bent je bewust van de kleine streepjes en van de mogelijke langere lijn. Je zou misschien denken “deze langere lijn is wat onzeker, hij is er *gedeeltelijk*”. Zulke informatie kan nog altijd expliciet voorgesteld worden, misschien met een feature type “onzekere lijn”. Voor de eenvoud zullen we veronderstellen dat de zekerheid op features geen continue schaal heeft (van 0 tot 1 bijvoorbeeld), maar gewoon iets discrete is, zoals “aan/uit”. Je kan enkel over entiteit X redeneren enkel en alleen als entiteit X aanwezig *is*, dus als het een bestaan heeft (ook als is dat denkbeeldig). Dus als je iets opgemerkt hebt aan de input, dan heb je het opgemerkt.

Laten we veronderstellen dat at runtime het aantal verschillende typen van features constant is. Aangezien het aantal pixels ook altijd constant is, is het totaal aantal features dat we kunnen ontdekken in de input langs boven begrensd door een (grote) constante.

4.3.1.1

4.3.2 Praktische opmerkingen

Features worden dus gebruikt om visuele informatie te modelleren, als een eerste stap naar interpretatie. De vraag blijft echter hoeveel features je moet vinden in de input. Men zou hiervoor een soort saliency map kunnen construeren en dan enkel features afleiden nabij gebieden in de saliency map van hoge activiteit. Er bestaan technieken die saliency maps en de “aandacht” van een computer programma op een visuele input proberen te modelleren, zoals [13]. Initieel zullen we ons hier echter niet mee bezig houden.

In het artikel [14] wordt voorgesteld dat een robot bij een input die hij moeilijk kan herkennen de zoom van de camera zou kunnen aanpassen en vervolgens meer gedetailleerde features kan afleiden. Maar dat gaat natuurlijk niet op foto’s. We leiden de features maar één keer af, we krijgen dus geen tweede kans. We kunnen wel heel veel mogelijke features afleiden, maar elke feature gaan we maar één keer afleiden. We moeten ons behelpen door zo intelligent mogelijke features af te leiden, maar deze hoeven niet allemaal verplicht stabiel te zijn onder ruis. Elke mogelijk feature die we afleiden is “iets” wat je kan opmerken aan de afbeelding.

In het artikel [10] wordt gezegd dat mensen afhankelijk van de taak ruwe vormen of gedetailleerde edges gebruiken in hun begrip van visuele informatie, bij het herkennen van visuele herinneringen. Een computer kan hier waarschijnlijk iets moeilijker flexibel mee omgaan. Een computer kan enkel de features matchen die hij heeft aangeleerd, onafhankelijk van het feit of die features ruwe visuele eigenschappen beschrijven of gedetailleerde. Het is ook gevaarlijk om de computer visuele herkenning *altijd* van coarse to fine scale te laten uitvoeren, zoals voorgesteld in [14], omdat soms de coarse information in een nieuwe visuele input soms helemaal niet in overeenstemming kan zijn met de aangeleerde coarse scale features. De computer matcht in het algemeen misschien beter op een niet-hiërarchische manier waarbij alle features even belangrijk zijn. Features omtrent ruwe visuele informatie zijn in principe even belangrijk als features omtrent gedetailleerde visuele informatie. Er is dus geen sprake van een *manier* of *strategie* waarop ons systeem bepaalde features verkiest boven andere. Welke features kenmerkend zijn voor een bepaalde entiteit zou gewoon moeten afhangen van een learning fase. Het is natuurlijk wel mogelijk dat gedetailleerde features enkel voorkomen bij een bepaalde unieke instantie en dat features van ruwe visuele informatie de enigsten zijn die uiteindelijk een categorie kunnen beschrijven zoals “koe” of “paard”. Elke koe heeft immers een ander concreet vlekken-patroon (gedetailleerde visuele informatie) dat zeer moeilijk bij een andere koe zouden zijn terug te vinden. De algehele vorm van een koe (ruwe visuele informatie) is wel sterk overeenkomstig tussen verschillende koeien.

Tot slot, het systeem dat we willen construeren heeft een geheugen en daarin zitten visuele herinneringen. Een visuele herinnering zouden we kunnen definiëren als een verzameling van features. Een visuele herinnering is zelf afkomstig uit de input van het gezichtsveld. Elk feature dat wordt gemaakt in het systeem krijgt een aparte identifier. Het heeft daarom geen zin om een feature tweemaal in de visuele herinnering te stoppen. Daarom is een herinnering een verzameling features en geen “bag” van features.

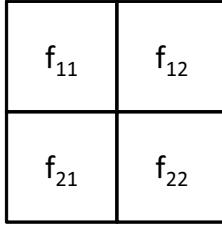
4.4 V1 gebaseerde piramide

In deze paragraaf beschrijven we onze kleine implementatie waarin enkele neuromodulatieve bevindingen uit paragraaf 4.1 worden uitgetest.

We hebben als input echte afbeeldingen gebruikt, maar de kleur-informatie hebben we niet benut: enkel de helderheid als een grijswaarde-schaal, met elke waarde in het interval $[0, 1]$.

4.4.1 Cellen die randen detecteren

In de artikels [5, 8, 15, 14] worden “Gabor filters” gebruikt om de simpele zenuwcellen in V1 te modelleren die randen moeten detecteren. De gedetecteerde randen worden dan verder geïntegreerd in contouren of wolken van losse edges. De Gabor filters in deze artikels hebben typisch een klein gebiedje van pixels nodig om op toegepast te worden als een lineaire filter. We hebben zelf een minimalistische versie bedacht om randen te detecteren. Neem een gebiedje van 2×2 pixels, een *quad* genaamd. Zie Figuur 4.5 voor de namen van de hokjes. In een dergelijke quad kan men vrij nuttige rand-detectie doen, zie Figuur 4.6 ter illustratie. Het is echter een zeer subjectief probleem om in een quad



Figuur 4.5: De namen van de hokjes in een quad van pixels.

van grijswaarden randen te detecteren omdat dit sterk beïnvloed wordt door de helderheid van de waarden en de relatieve contrasten. Het is bovendien mogelijk dat er meerdere randen in één quad kunnen gedetecteerd worden, afhankelijk van de subtile structuur van de grijswaarden.

Eerst hadden we hiervoor een aantal simpele regeltjes geprogrammeerd, bijvoorbeeld:

Horizontale rand indien

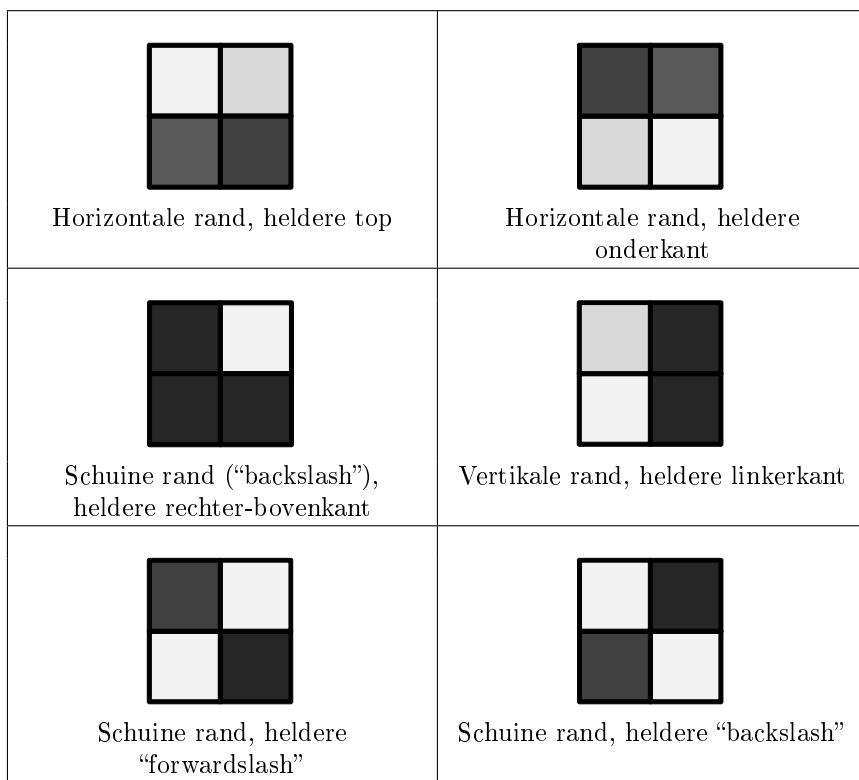
```
(isDarker(f11, f21) AND isDarker(f12, f22)) OR  
(isBrighter(f11, f21) AND isBrighter(f12, f22))
```

met

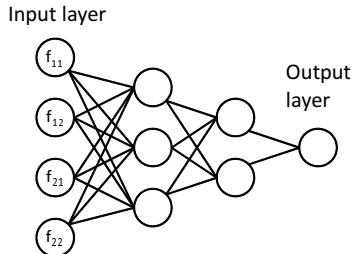
$$\begin{aligned} \text{isDarker}(a,b) &:= |a - b| \geq T \text{ AND } a < b \\ \text{isBrighter}(a,b) &:= |a - b| \geq T \text{ AND } a > b \end{aligned}$$

Hierbij is T een threshold die aangeeft wanneer twee grijswaarden perceptueel verschillend zijn. Dit soort regeltjes zijn echter veel te eenvoudig. De quads waarvan we zouden vinden dat zij een horizontale rand bevatten worden zeker als zodanig bestempeld, maar ook heel veel quads die perceptueel geen horizontale rand bevatten worden als horizontale rand bestempeld. Waarschijnlijk is de threshold T ook iets te ruw omdat het feit of je twee grijstinten kan onderscheiden van elkaar afhangt van hoe helder deze tinten zijn (een soort logaritmisch verband).

Omdat het herkennen van randen redelijk subjectief is en het verzinnen van harde regeltjes voor hun detection niet echt eenvoudig is, zijn we in het machine learning boek van Tom Mitchell [7] op zoek gegaan naar algoritmen. De neurale netwerken uit hoofdstuk 4 leken ons interessant omdat zij reële getallen als input nemen. We kunnen dus de waarden in de quad als input aanbieden. We gaan niet in op de details en verwijzen door naar het geraadpleegde werk voor meer informatie. Het is mogelijk om de gewichten te trainen op verschillende manieren. Wij hebben gekozen voor de *backpropagation* manier. In [7] wordt opgemerkt dat er geen echte design principes zijn om het aantal knopen en aantal layers in een neurale netwerk te bepalen, maar dat er sowieso genoeg interne knopen aanwezig moeten zijn om “interne voorstellingen” te berekenen. Teveel knopen leidt tot een trage convergentie van de gewichten en langere leertijd. Er is ook een belangrijke parameter die men dient in te stellen, de *learning rate*. De



Figuur 4.6: Voorbeelden van randen die je kan detecteren als je slechts 2×2 pixels ter beschikking hebt.



Figuur 4.7: De structuur van de gebruikte neurale netwerken

learning rate bepaalt hoe sterk het neurale netwerk moet aangepast worden per training sample. Dit mag men niet te hoog instellen, want anders convergeert het netwerk niet. Wij hebben gewerkt met een arbitraire waarde 0.25.

Wij hebben voor elk type rand dat we wilden detecteren een apart neurale netwerk gemaakt, met de structuur zoals getoond in Figuur 4.7. De output die men verwacht op een gegeven input hangt af van de manier waarop men training samples opstelt. We hebben bijvoorbeeld volgende soort training data voor een horizontale rand met een heldere bovenkant:

```
0.78 0.78 0.21 0.21; 1.0
0.46 0.46 0.18 0.18; 1.0
0.41 0.27 0.12 0.28; 0.0
0.25 0.37 0.43 0.18; 0.0
```

Elke regel stelt één training sample voor. De vier reële getallen stellen de quadwaarden $f_{11}, f_{12}, f_{21}, f_{22}$ voor in volgorde. Het vijfde getal stelt “true” (1.0) of “false” (0.0) voor. Elk training sample is dus in essentie een concrete quad van waarden en daarbij het juiste antwoord. De training samples per type rand worden wel apart gehouden.

Omdat een neurale netwerk reële waarden teruggeeft moeten qualitatieve labels ook met reële getallen worden gemodelleerd. Voor elk type rand, waaronder de types van Figuur 4.6, hebben we veel van dergelijke training samples gegenereerd. Dit werd gedaan door random quads te genereren, te presenteren aan de gebruiker met een ja-neen vraag zoals in Figuur 4.8.

Als eenmaal een neurale netwerk is bekomen voor een edge type, zal de output op een willekeurige input quad nooit netjes 1.0 of 0.0 zijn. De waarden van de enige output-knoop liggen verspreid over de range [0, 1]. Om de output-waarde interpreteren als true of false hebben we een thresholding-strategie gebruikt: als de output-waarde bijvoorbeeld boven 0.7 ligt, dan is het antwoord true, anders false. Ook werd na het trainen van het neurale netwerk gekeken op welke quads het neurale netwerk een foutief antwoord gaf en dit werden dan nieuwe training samples (met het juiste antwoord dan wel).

De resulterende neurale netwerken zijn over het algemeen strenger dan de simpele regels die we eerst hadden geïmplementeerd, hoewel de nauwkeurigheid natuurlijk niet optimaal is. Wanneer een neurale netwerk van een bepaald edge type “true” output op een input quad, komt dit antwoord vaak overeen met de eigen subjectieve perceptie.



Figuur 4.8: De gebruiker kan aangeven of een quad voldoet aan het edge type onder beschouwing of niet. Hier wordt concreet de vraag gesteld of de getoonde quad een horizontale rand voorstelt met een heldere bovenkant.

4.4.2 De piramide

De artikels die aan het begin van paragraaf 4.4.1 genoemd zijn werken ook steeds vanuit het idee dat de processing in het V1 gebied van de hersenen hiërarchisch gebeurt: onderaan heb je simpele cellen die randen detecteren, hogerop heb je complexere cellen die deze randen integreren. We hebben nagedacht over deze manier van werken, maar hebben het idee van de artikels niet letterlijk overgenomen. We hebben een eigen piramide uitgewerkt die enkel gebruik maakt van de simpele cellen zelf en de manier waarop deze elkaars signalen tegenwerken (inhibition en facilitation uit paragraaf 4.1). De complexe cellen die het maximum-signaal selecteren uit de onderliggende simpele cellen zijn in onze implementatie impliciet aanwezig en hebben we niet expliciet gemodeerdeerd. We bespreken hieronder hoe de “signalen” uit de simpele cellen worden samengenomen tot een nieuwe abstractie.

We nemen als input een image. We zetten deze om naar een grijswaarde image omdat zoals gezegd we geen kleurinformatie gebruiken. Voor de eenvoud is de hoogte van een input image gelijk aan de breedte w en beiden zijn een macht van twee. We delen de image op in quads. Als we de posities van $f_{11}, f_{12}, f_{21}, f_{22}$ voorstellen als een vier-tuple van (x, y) paren, dan worden de quads gegeven door verzameling

$$\{(x, y), (x + 1, y), (y + 1, x), (x + 1, y + 1) \mid x, y \in [0, w - 1] \cap \mathbb{N}\}$$

Elke quad wordt door de neurale netwerken van de alle edge types bekeken en de bijhorende edge types worden al dan niet gedetecteerd in elke quad. Elke quad wordt op die manier omgezet naar een bijhorende verzameling edges. Alle edges liggen dus in een gridstructuur omdat de quads zelf in een grid-structuur liggen. Ook al hebben we slechts vier richtingen voor onze edges, we weten dat we deze edges aan elkaar kunnen koppelen over meerdere quads heen om zachtere contouren te vormen.

Vervolgens gaan de edges inhibition en facilitation toepassen op elkaar. Hiervoor wordt een *surround-model* gebruikt dat door de gebruiker zelf kan geconstrueerd worden in een editor zoals getoond in Figuur 4.9. Het concept van cell

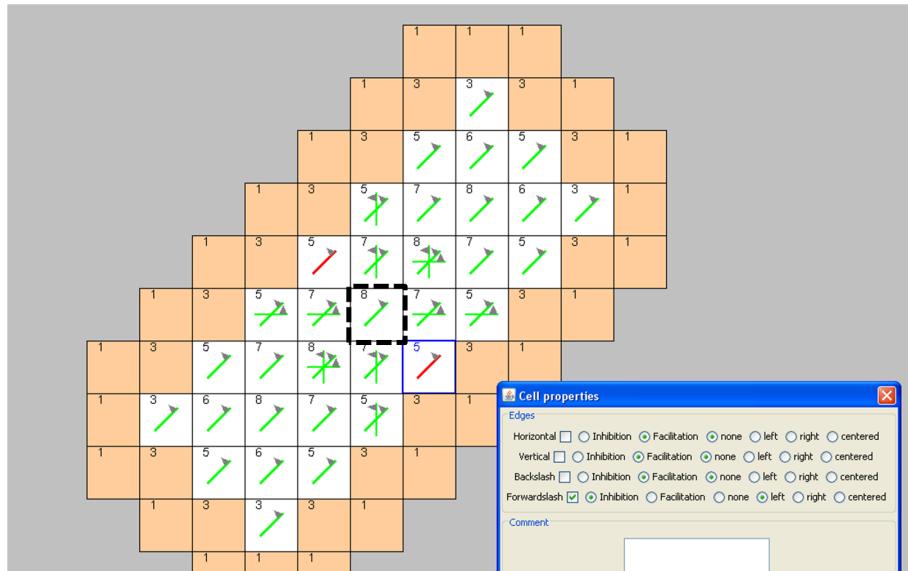
surround is kort geïntroduceerd in paragraaf 4.1. We blijven door deze editor dus data-driven en de gebruiker kan zelf beslissen hoe ver de invloed van een edge reikt en welke edges erdoor beïnvloed worden om andere manieren van contour integration te bestuderen. Elk surround-model heeft een centrum-hokje. Hierin mag slechts één edge type worden ingesteld. Wanneer we inhibition en facilitation toepassen op de edges bekomen uit de input images, wordt het centrum-hokje van een surround-model gecentreerd boven elke quad. We doen dit enkel indien de edge in dit centrum-hokje effectief werd gedetecteerd in de quad. De andere hokjes van het surround-model vormen dan de invloedsomgeving van de edge waarop we gecentreerd hebben.

Voor elke edge houden we een tellertje bij. Dit tellertje wordt verhoogt wanneer een edge in zijn surround facilitation toepast op deze edge en verlaagd indien een edge in zijn surround inhibition toepast. Nadien worden *dominante edges* overgehouden waarvoor de waarde van hun tellertje boven een bepaalde threshold uitkomt.

Nu gaan we een stap hoger in de piramide, waar we opnieuw hetzelfde herhalen, maar dan op een kleinere versie van de vorige input image (één vierde zo groot als de vorige). De dominante edges mogen stemmen voor de grijswaarden in de input-image op een niveau hoger. Een dominante edge stemt feitelijk gewoon voor de waarden in de quad waarin hij werd gedetecteerd. De waarden van de quads krijgen dan gewichten en bij het downsamplen zullen de waarden met de hoogste gewichten sterker bewaard blijven. Hierdoor krijgen we afhankelijk van de hoeveelheid dominante edges een nieuwe image waarin bepaalde randen geaccentueerd zijn.

4.4.3 Een voorbeeld

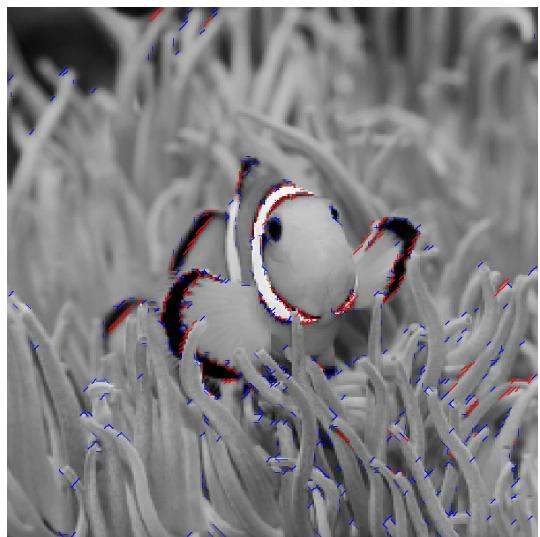
We tonen hier enkele voorbeelden die het effect van onze eenvoudige V1-implementatie tonen. Beschouw de originele afbeelding in Figuur 4.10. Deze hebben we gevonden op de website www.photoforum.ru. In Figuur 4.11 tonen we de afbeelding geproduceerd door de verwerking van de eerste laag simpele cellen. Deze afbeelding heeft minder resolutie omdat we voor elke quad enkel één nieuwe grijswaarde onthouden waarin de waarden van de cellen met de meest dominante randen het meeste aandeel hebben. De dominante randen zijn in het rood weergegeven. In het blauw zijn de niet-dominante randen aangegeven. Er zijn ook veel plaatsen waar ons model geen randen heeft gedetecteerd omdat daar de visuele randen zijn “verspreid” over meerdere quads heen waardoor we per quad moeilijk een rand kunnen opmerken. In Figuur 4.12 tenslotte tonen we de geproduceerde afbeelding na het vierde niveau van simpele cellen. Men kan zien dat de gebieden waar in Figuur 4.11 sterke dominante randen aanwezig waren hun oorspronkelijke grijswaarden en contrasten redelijk sterk hebben behouden.



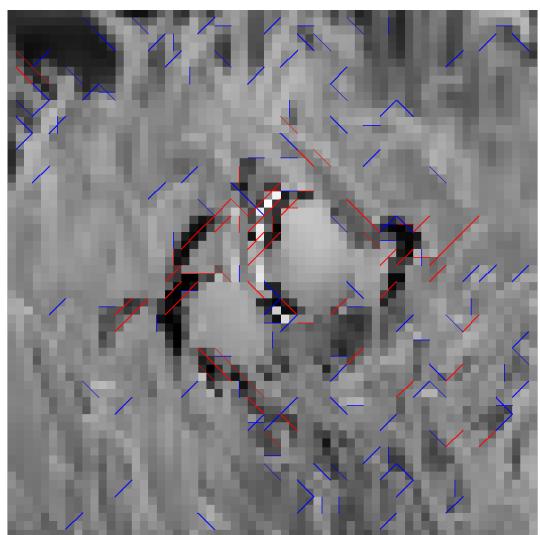
Figuur 4.9: Een editor waarin de gebruiker kan een cell surround model maken dat aangeeft hoe de verschillende edge types elkaar signalen kunnen versterken (facilitation, getoond in groene kleur) of tegenwerken (inhibition, getoond in rode kleur). Elk hokje stelt een aparte cell voor. De getallen in een hokje geven gewoon aan hoeveel opgevulde buur-cellen hij heeft. Voor elk hokje kan apart een dialoog-venster opgeroepen worden om de instellingen te doen. Het dialoog venster in de figuur hoort bij het hokje met een dun blauw randje. Het centrum van het model is getoond in een zwarte stippe lijn.



Figuur 4.10: Originele afbeelding (omgezet naar grijswaarden)



Figuur 4.11: Verwerkte afbeelding



Figuur 4.12: Verwerkte afbeelding

Hoofdstuk 5

Future work

We hebben nog een aantal concrete voorstellen voor toekomstig werk.

We moeten nader onderzoeken op welke manier ons frequent set intersection mining algoritme te vergelijken valt met het zoeken naar large itemsets. Het lijkt erop dat beide problemen equivalent zijn en in dat geval is het interessant om te kijken of ons algoritme bepaalde nieuwe kwaliteiten heeft ten opzichte van de reeds bestaande literatuur over dit onderwerp.

In de context van ons frequent subgraph mining algoritme zou het ook interessant zijn om te onderzoeken of in bepaalde omstandigheden of toepassingen het voldoende zou zijn om slechts een fractie van alle mogelijke paarsgewijze matchings uit te rekenen. Merk ook op dat we niet verplicht zijn om het MCIS probleem op te lossen om deze matchings te vinden. Het is niet ondenkbaar dat men via snellere en eventueel zelfs polynomiale algoritmen ook matchings kan bekomen die, hoewel ze misschien weliswaar een benadering zijn van de exacte matchings, toch voldoende “groot” zijn om de frequent set intersection mining uit te voeren op hun uiteinden. Bovendien zijn we ook niet verplicht het frequent set intersection mining probleem exact en exhaustief op te lossen. Misschien is een greedy intersectie-manier van de verzameling bij sommige toepassingen ook al in staat veel interessante patronen te ontdekken.

We willen uiteraard ook de rest van onze visie geschetst in hoofdstuk 1 verder blijven uitwerken en een geheel werkend systeem implementeren dat binnen het domein van de (visuele) patroonherkennung competitief kan zijn met de technieken uit de bestaande literatuur. We vermoeden dat het modelleren van neurobiologische en psychologische bevindingen omtrent de werking van de menselijke hersenen met technieken en formalismen uit de theoretische informatica waarschijnlijk in de (nabije) toekomst steeds belangrijker en belangrijker zal worden. De nood aan flexibeler manieren om informatie te verwerken laat zich bijvoorbeeld sterk voelen in het omgaan met multimedia content. Helaas is dit geen eenvoudig probleem en daarom moeten we rekening houden met veel aspecten.

Bibliografie

- [1] T. Washio A. Inokuchi and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD'00*, pages pages 13–23, 2000.
- [2] David R. Andresen, Joakim Vinberg, and Kalanit Grill-Spector. The representation of object viewpoint in human visual cortex. *NeuroImage*, 45:522–536, 2009.
- [3] Steven C. Dakin and Nina J. Baruch. Context influences contour integration. *Journal of Vision*, 9(2):1–13, 2009.
- [4] Rudy Guyonneau, Holle Kirchner, and Simon J. Thorpe. Animals roll around the clock: The rotation invariance of ultrarapid visual processing. *Journal of Vision*, 6(10):1008–1017, 2006.
- [5] Tim C. Kietzmann, Sascha Lange, and Martin Riedmiller. Computational object recognition: a biologically motivated approach. *Biological Cybernetics*, 100(1):59–79, February 2009.
- [6] Qi Li and Haiyan Geng. Progress in cognitive neuroscientific studies of visual awareness. *Progress in Natural Science*, 19:145–152, 2009.
- [7] Tom M. Mitchell. *Machine Learning*. McGraw-HILL INTERNATIONAL EDITIONS Computer Science Series, 1997.
- [8] Jim Mutch and David G. Lowe. Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision*, 80(1):45–57, 2008.
- [9] Jeffrey Ng, Anil A. Bharath, and Li Zhaoping. A survey of architecture and function of the primary visual cortex (V1). *EURASIP Journal on Advances in Signal Processing*, 2007:Article Number 97961, 2007.
- [10] A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in Brain Research: Visual perception*, 155:23–36, 2006.
- [11] Bruno A. Olshausen and David J. Field. How close are we to understanding V1? *NEURAL COMPUTATION*, 17(8):1665–1699, 2005.
- [12] Marius V. Peelen, Li Fei-Fei, and Sabine Kastner. Neural mechanisms of rapid natural scene categorization in human visual cortex. *Nature*, 460(7251):94–U105, JUL 2009.

- [13] Tomaso Poggio, Thomas Serre, Cheston Tan, and Sharat Chikkerur. An integrated model of visual attention using shape-based features, JUN 2009.
- [14] J Rodrigues and J. M. Hans du Buf. A cortical framework for invariant object categorization and recognition. *COGNITIVE PROCESSING*, 10(3):243–261, AUG 2009.
- [15] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *Proceedings of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005)*. IEEE Computer Society Press, San Diego, 2005.
- [16] Guy Wallis, Benjamin T. Backus, Michael Langer, Gesche Huebner, and Heinrich Bülthoff. Learning illumination- and orientation-invariant representations of objects through temporal association. *Journal of Vision*, 9(7):1–8, 2009.
- [17] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *IEEE INTERNATIONAL CONFERENCE ON DATA MINING, PROCEEDINGS*, 2002.
- [18] Li Zhaoping and Peter Dayan. Pre-attentive visual selection. *NEURAL NETWORKS*, 19(9):1437–1439, NOV 2006.