Dankwoord

Het resultaat van deze thesis had nooit tot stand kunnen komen zonder een aantal mensen die mij hierin rechtstreeks en onrechtstreeks bijstonden. In de eerste plaats wil ik vooral mijn promotor Prof. Dr. Bart Kuijpers en mijn begeleidster Sofie Haesevoets bedanken voor de goede begeleiding doorheen het hele jaar. Verder wil ik ook mijn ouders bedanken voor hun steun doorheen mijn opleiding en de kans die ze mij gaven om te studeren. Mijn zus, van wie ik tot vervelens toe de computer heb mogen lenen om wat extra testen uit te voeren, en de rest van de familie. Mijn vrienden waarmee ik altijd het nodige plezier kon beleven om de moed erin te houden gedurende mijn studies. De leidingsploeg van de Scouts voor de steun en het af en toe moeten missen van hun groepsleider. En natuurlijk ook mijn medestudenten op het LUC gedurende de volledige vier jaar.

Bedankt allemaal!

Ward Jans.

Abstract

Het doel van deze thesis is om spatial en spatio-temporal data mining te bestuderen en toe te passen in de verkeerskunde. In hoofdstuk 2 worden eerst een aantal mogelijke datamodellen besproken, deze datamodellen zullen verder gebruikt worden om op te minen. Vervolgens is er nood aan queries om over deze datamodellen te stellen, deze worden besproken in hoofdstuk 3. In hoofdstuk 4 worden eerst nog een aantal begrippen beschreven die nodig zijn alvorens met spatial en spatio-temporal mining van start te kunnen gaan. In hoofdstukken 5 en 6 worden respectievelijk spatial en spatio-temporal mining algoritmes besproken.

Op het gebied van spatio-temporal data mining is nog niet zoveel onderzoek gebeurd en zijn er dan ook nog geen concrete toepassingen in de verkeerskunde. Er is dus nood aan nieuwe ideeën. In hoofdstsuk 7 worden bepaalde wetmatigheden in verband met verkeer besproken, en hierop gebaseerd worden nieuwe ideeën uitgewerkt voor spatio-temporal data mining in de verkeerskunde. Deze worden ook geïmplementeerd en worden beschreven en getest in hoofdstuk 8. Ze worden in dat hoofdstuk ook met elkaar vergeleken.

In hoofdstuk 9 worden de uiteindelijke eindresultaten van de thesis samengevat. Deze zullen hoofdzakelijk betrekking hebben op hoofdstukken 7 en 8.

Inhoudsopgave

1 Inleiding **4**

2 Datamodellen **5**
2.1 Point Location Management 5
2.2 Trajectory Location Management 6
2.2.1 Datamodel 6
2.2.2 Voordelen ten opzichte van het Point Location Management 7
2.3 Linear Constraint Model 9
2.3.1 Moving Points 9
2.3.2 Het datamodel 11
2.4 Uncertainty in het trajectory location management datamodel 13

3 Queries **17**
3.1 Trajectory Location Management Queries 17
3.1.1 Soorten queries 17
3.2 Queries op het Linear Constraint Model 22
3.2.1 Query-taal 22
3.2.2 Kwantor-eliminatie 25

4 Neighborhood-begrippen **26**
4.1 Neighborhood-relaties 26
4.1.1 Topologische relaties 26
4.1.2 Afstandsrelaties 27
4.1.3 Richtingsrelaties 28
4.2 Neighborhood-grafen en hun operaties 29
4.2.1 Neighborhood-grafen en paden 29
4.2.2 Enkele operaties 30
4.3 Neighborhood-indices 31

5 Spatial Mining **35**
5.1 Spatial Patronen 35
5.2 Algoritmes voor Spatial Mining 36
5.2.1 Efficiënte klasse identificatie 36
5.2.2 Spatial characterization 41
5.2.3 Spatial Trend Detection 44
5.2.4 Efficiënte clustering met noise removal 49
5.2.5 Mining naar spatial association rules 56

6 Spatio-temporal Mining **61**
6.1 Spatio-Temporal Patronen 61
6.2 Spatio-temporal clustering 61
6.3 Effeciënte mining naar bewegingspatronen 64
6.3.1 De gebruikte bewegingspatronen 64
6.3.2 Het flock-algoritme 66
6.3.3 Het leadership-algoritme 68
6.3.4 Het convergence-algoritme 69
6.3.5 Het encounter-algoritme 70
6.4 Een clustering aanpak voor het ontdekken van persoonlijke geografische indexen 72
6.4.1 Eerdere pogingen 72
6.4.2 DJ-clustering 74

7 Toepassing in de verkeerskunde **77**
7.1 Het domein verkeer en de problemen 77
7.2 Gedrag van verkeer en verkeersstromen 78
7.3 Waar data mining kan helpen 80
7.4 Data mining na de feiten 81
7.5 Real time data mining 84

8 Implementatie **87**
8.1 Het gebruikte model 87
8.2 Technieken en testen 89
8.2.1 De gebruikte bron-bestemmingsmatrices 89
8.2.2 De technieken 90
8.2.3 Testen 91
8.3 Resultaten 91
8.3.1 Enkel traject Antwerpen - Charleroi 91
8.3.2 Toestroom naar Brussel 105
8.4 Implementatie- en testgegevens 111

9 Conclusie **112**

Hoofdstuk 1

Inleiding

Het is waarschijnlijk niet moeilijk om enkele problemen op te sommen die zich zoal voordoen in het verkeer. Iedereen zal al wel ooit eens in de file gestaan hebben, of hinder ondervonden hebben van een verkeersongeval. Het is daarom ook niet nutteloos hiervoor mining technieken te bedenken die deze problemen aanpakken. Er wordt vandaag de dag wel al gebruik gemaakt van filemeldingen via onder andere radio, maar het is wenselijk om veel betere technieken ter beschikking te hebben om files te mijden.

Een auto die over een weg rijdt, heeft een bepaalde locatie op een bepaald tijdstip. Deze locatie verandert doorheen de tijd vermits de auto zich verplaatst over de weg. Rijdende auto's kunnen dus goed gemodelleerd worden als spatio-temporal objecten. Hiervoor bestaan reeds een aantal modellen die gebruikt kunnen worden.

Het zou zeer aangenaam zijn moesten routeplanners in combinatie met een GPS-systeem simpelweg rekening kunnen houden met druktes en dergelijke. Een chauffeur zou dan, als hij wil vertrekken, simpelweg zijn eindbestemming moeten ingeven en het systeem doet de rest. Het systeem moet dan alle data van alle auto's bijhouden en zo heel het verkeer regelen.

Om dit te realiseren kan data mining van groot nut zijn. Zoals gezegd kunnen auto's worden voorgesteld als spatio-temporal objecten en kunnen er spatio-temporal data mining technieken gebruikt worden om hierop te minen. In het verloop van de thesis zal echter blijken dat de bestaande technieken hiervoor niet echt geschikt zijn en dat er nieuwe ideeën nodig zullen zijn.

In hoofdstukken 2 tot en met 6 worden datamodellen, queries hierop en gewone spatial en spatio-temporal data mining algoritmes besproken. De nieuwe ideeën met betrekking tot de verkeerskunde en de uitwerkingen hiervan worden in hoofdstukken 7 en 8 besproken. De thesis eindigt met een korte samenvatting van de bekomen resultaten.

Hoofdstuk 2

Datamodellen

Om te beginnen zal in dit hoofdstuk een eerste belangrijk aspect van spatial en spatio-temporal dataminig behandeld worden, namelijk het te gebruiken datamodel. Zijn de bestaande relationele datamodellen bruikbaar voor spatial en spatio-temporal data? Is er nood aan nieuwe modellen? Het is waarschijnlijk niet moeilijk in te zien dat een gewone relationele database hiervoor totaal niet geschikt is. In dit hoofdstuk worden drie verschillende datamodellen voorgesteld: het *Point Location Management datamodel*, het *Trajectory Location Management datamodel* en het *Linear Constraint datamodel*.

2.1 Point Location Management

Een eerste mogelijkheid die bedacht kan worden voor het voorstellen van moving object data is het systematisch opslaan van tijd-locatie tupels van de vorm (l, t), waarbij l de coördinaten zijn van het moving object en t het tijdstip waarop het moving object zich op locatie l bevindt. Dit systeem heet het *Point Location Management* systeem. Dit systeem is bijvoorbeeld terug te vinden in transport management systemen waar men wil kunnen uitdrukken op welke plaatsten en op welke tijdstippen een vrachtwagen levert. Dit systeem, vermeld door Wolfson in [Wol02], is duidelijk een naïeve poging om moving objects te modelleren. Er zijn aan dit systeem enkele nadelen verbonden. Een eerste nadeel is dat er geen inter- of extrapolatie mogelijk is omdat de enige beschikbare gegevens tijd-plaats tupels op discrete tijdstippen zijn. Dit zorgt ervoor dat er niets geweten is van waar het object zich situeert tussen twee op elkaar volgende tijdstippen. Een volgend nadeel is dat er een grote trade-off is tussen de precisie en de benodige opslagruimte. Om een goede precisie te hebben moet op veel tijdstippen de locaties van de verschillende objecten worden opgeslagen maar dat moet dan

allemaal ingeboet worden aan opslagruimte. Om weinig opslagruimte te verbruiken moet er dan weer gesnoeid worden in het aantal opgeslagen locaties wat zorgt voor een slechte nauwkeurigheid. Een laatste nadeel is dat de ontwikkeling van software die dit systeem hanteert zeer inefficiënt werkt. Dit probleem is te wijten aan drie belangrijke redenen. Ten eerste zijn bestaande database management systemen er niet op voorzien om continu wijzigende data te verwerken. Ten tweede is SQL niet ontwikkeld om met informatie over tijd en plaats om te gaan en ten derde maakt het feit dat de exacte locaties van moving objects niet juist geweten zijn het heel moeilijk om te kunnen indexeren en queries te stellen. Omwille van deze nadelen wordt dit systeem slechts in enkele specifieke gevallen gebruikt. Deze gevallen kunnen grosso modo opgedeeld worden in twee categorieën: moving objects waarvan de route op voorhand bekend is (vb. transport management) en moving object waarvan er niet meer informatie beschikbaar is dan tijd-locatie koppels (vb. de vijand in digital battlefield applicaties). Er is dus duidelijk nood aan betere systemen die deze nadelen niet hebben. Deze systemen zullen in de volgende paragrafen aan bod komen.

2.2 Trajectory Location Management

Wolfson stelt in [Wol02] ook vast dat het Point Location Management systeem weinig praktisch nut blijkt te hebben. Er wordt dan ook een model voorgesteld dat niet meer te kampen heeft met al deze nadelen. Dit model heet het *Trajectory Location Management* systeem. Het idee achter dit model is dat de routes of trajecten van de verschillende objecten worden opgeslagen.

2.2.1 Datamodel

Een traject van een moving object wordt bepaald door het startpunt, het tijdstip waarop het moving object zich op dit startpunt bevindt en het eindpunt. Om het effectieve traject voor te stellen wordt er gebruik gemaakt van lijnsegmenten. Een traject is een aaneenschakeling van lijnsegmenten. Zij o een moving object dan is $P(o)$ de sequentie lijnsegmenten die het kortste pad, in de veronderstelling dat hiervoor een algoritme beschikbaar is, van het startpunt naar het eindpunt voorstelt. $L(o)$ is de polyline die de effectieve aaneenschakeling is van de verschillende lijnsegmenten van $P(o)$. Een polyline voorgesteld in 2D bijvoorbeeld wordt bepaald door een opsomming van x, y coördinaten: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$. Vermits $P(o)$ een aaneenschakeling is van lijnsegmenten die het korste pad vormen tussen start- en eindpunt, moet het eindpunt van elk segment steeds samenvallen met het

beginpunt van het volgende segment en kan elke $P(o)$ dus uitgedrukt worden als een polyline. Met al deze gegevens kan vervolgens een *certain-trajectory* (of *c-trajectory*) geconstrueerd worden. Een c-trajectory is van de vorm:

$$(x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n)$$

Op deze manier wordt aangegeven dat een moving object een weg aflegt van het (start-)punt (x_1, y_1) naar het (eind-)punt (x_n, y_n) vertrekkende van op het startpunt op tijdstip t_1 en aankomend op het eindpunt op tijdstip t_n. Om de tijden te kunnen berekenen waarop het moving object zich op de verschillende punten bevindt, wordt er gebruik gemaakt van de *DriveTime*. DriveTime is de theoretische tijd die nodig is om van het startpunt van een lijnsegment tot aan het eindpunt te geraken. Door systematisch de tijd bij te tellen die nodig is om het lijnsegment af te leggen, kan het theoretische tijdstip van aankomst berekend worden. Uiteraard introduceert het gebruiken van deze theoretische DriveTime een kleine afwijking, maar hier wordt uitgebreid op teruggekomen in sectie 2.4.

Een database die een verzameling moving objects bevat, wordt een Moving Objects Database (MOD) genoemd.

Voorbeeld 2.1. Zij D een MOD met twee objecten o_1 en o_2. Zij c_1 en c_2 twee c-trajectories van respectievelijk o_1 en o_2:

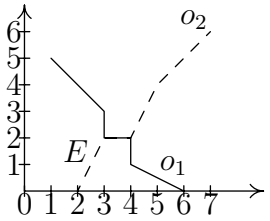
$$c_1 : (1, 5, 0), (3, 3, 1), (3, 2, 2), (4, 2, 3), (4, 1, 4), (6, 0, 5)$$
$$c_2 : (2, 0, 0), (3, 2, 2), (4, 2, 3), (5, 4, 4), (7, 6, 5)$$

Dan kunnen de trajecten van de twee objecten grafisch worden weergegeven zoals getoond in figuur 2.1. Beide objecten vertrekken op tijdstip 0. Na twee tijdseenheden komen ze samen op het punt $(3, 2)$, en vervolgens nemen ze eenmaal hetzelfde lijnsegment. Daarna scheiden hun trajecten opnieuw. Merk op dat o_2 lang over het segment E doet. Als o_1 en o_2 twee auto's zouden zijn, dan zou E bvb. een drukke straat kunnen zijn.

□

2.2.2 Voordelen ten opzichte van het Point Location Management

Uiteraard zijn er aan dit model een aantal voordelen verbonden ten opzichte van het point location management. Het trajectory location management laat wel inter- en extrapolatie toe. Wanneer een moving object een lijnsegment volgt van t_1 tot t_2, dan zal het object zich op tijdstip $\frac{t_1+t_2}{2}$ exact, de



Figuur 2.1: Routes van objecten o_1 en o_2

afwijking buiten beschouwing gelaten, in het midden van het lijnsegment bevinden.

Het inboeten van benodige opslagruimte aan precisie is in dit model ook niet meer het geval. Om in het point location management model te weten waar een object zich bevindt op een bepaald tijdstip moet dit expliciet opgeslagen worden in de database. Bij het trajectory location management model kan de locatie van een object berekend worden zonder dat hiervoor extra ruimte nodig is.

Om het derde probleem op te lossen van het point location management systeem, nl., inefficiënte software-ontwikkeling moeten er exta operatoren worden toegevoegd. Deze operatoren kunnen opgedeeld worden in een aantal categorieën en worden hier kort vermeld.

Operatoren die toegevoegd worden met betrekking tot een enkel traject:

- *WHEN object* `o` *CLOSEST TO address* `x`: Deze operator geeft een lijst terug van tijdstippen waarop het object `o` op locatie `x` is. Indien het object nooit op de locatie komt, wordt de locatie berekend waarop het object het dichtst bij de gevraagde locatie is, dit kan ook op meerdere tijdstippen zijn.

- *VCR object* `o`: Deze operator speelt het traject van object `o` opnieuw af op een bepaalde tijdsschaal bvb. 1 seconde per minuut. Het is ook mogelijk om snel vooruit of achteruit te gaan (vergelijkbaar met echte VCR dus).

Operatoren op trajecten die in relatie staan tot een gebied of punt:

- $[always|sometime][somewhere|everywhere][possibly|definitely]$ *inside* (T, R, t_1, t_2): T is het traject van het moving object, R is het gebied en $[t_1, t_2]$ is het interval waarop de query betrekking heeft. Omdat de locatie van een moving object continu veranderd kunnen er drie verschillende vragen gesteld worden t.o.v een gebied:

  - Is het object altijd of ooit in het gebied?
  - Is het object ergens of overal in het gebied?
  - Is het mogelijk of zeker in het gebied? (zie sectie 2.4)

  In het totaal zijn er dus $2^3 = 8$ verschillende operatoren mogelijk.

Operatoren met betrekking tot relaties tussen trajecten:

- *Possibly-Within*[*distance d*|*travel-time t*]*sometime in the time interval T*: Deze condities zijn waar voor koppels van trajecten die in het tijdsinterval T op afstand d of tijd t van elkaar verwijderd zijn.
- *Possibly-Fartherthan*[*distance d*|*travel-time t*]*sometime in the time interval T*: Omgekeerde operator van de vorige.

Door de toevoeging van deze operatoren is het probleem opgelost dat er geen efficiënte software ontwikkeld kan worden om met moving objects om te gaan. Het gebruikte datamodel zorgt ervoor dat database management systemen nu wel continu wijzigende data aankunnen. Het toevoegen van de operatoren zorgt ervoor dat de data wel ondervraagd kan worden en het berekenen van de exacte locaties van objecten wordt ook mogelijk gemaakt door extrapolatie.

2.3 Linear Constraint Model

Een ander datamodel om moving objects te modelleren in een database is het *Linear Constraint model* voorgesteld door Su, Xu en Ibarra ([SXI01]). Het idee is hier om een moving object voor te stellen aan de hand van een continue en oneindig afleidbare functie $f : time \rightarrow \mathbb{R}^n$, waarbij het domein *time* alle reële getallen bevat.

2.3.1 Moving Points

In deze sectie zal het begrip moving points worden gedefinieerd. Het volledige datamodel wordt voorgesteld in sectie 2.3.2

Definitie 2.1. Een *moving point* is een functie **p**: time $\rightarrow$ $point_n$ die voorgesteld kan worden als een vector van functies $\mathbf{p}(t) = (p_1(t), p_2(t), ..., p_n(t))$ en voldoet aan volgende condities:

- p_i is continu over de tijd voor elke $1 \leq i \leq n$

- Er bestaat een getal m en m tijdstippen $t_1 < t_2 < \ldots < t_m$ zodanig dat:
  - $p_i|_{(-\infty,t_1)}$ en $p_i|_{(t_m,+\infty)}$ oneindig afleidbaar zijn, en
  - voor elke $1 \leq j \leq m-1$, $p_i|_{(t_j,t_{j+1})}$ ook oneindig afleidbaar is.

□

Voorbeeld 2.2. Beschouw een vliegtuig dat rondvliegt in een driedimensionale ruimte voorgesteld als een moving point als volgt:

$$\begin{aligned} \mathbf{p}(t) &= (2t-40, -t+23, 30) \wedge 0 \leq t \leq 21 \\ \vee\, \mathbf{p}(t) &= (2, -t+23, -5t+135) \wedge 21 \leq t \leq 22 \\ \vee\, \mathbf{p}(t) &= (0.5t-9, 1, -t+47) \wedge 22 \leq t \leq 47 \\ \vee\, \mathbf{p}(t) &= (14.5, 1, 0) \wedge 47 \leq t \end{aligned}$$

Het vliegtuig vliegt eerst in zuid-oostelijke richting en draait op tijdstip 21 en begint ook te dalen. Op tijdstip 22 maakt het opnieuw een draai en blijft het dalen. Het vliegtuig zal uiteindelijk landen (z = 0) op tijdstip 47 en blijven stilstaan. Alle p_i's $(2t-40, -t+23, 30, 2, -t+23, ...)$ zijn oneindig afleidbaar en continu. Merk op dat de verschillende intervallen aansluiten. Met andere woorden de locatie van het vliegtuig berekend in het eindpunt van het ene interval is hetzelfde als de locatie berekend op het beginpunt van het volgende interval. Op tijdstip 21 bevindt het vliegtuig zich op het punt $(2, 2, 30)$, op het tijdstip 22 op het punt $(2, 1, 25)$ en het landt uiteindelijk op het punt $(14.5, 1, 0)$. □

Door het gebruik van vectoren in dit datamodel is het makkelijk om *velocity* (snelheidsvector) te definiëren als $vel(\mathbf{p}) = \mathbf{p}' = (p_1', p_2', \ldots, p_n')$ (of eventueel genormaliseerd als $\frac{vel(p)}{||vel(p)||}$). De *speed* (getalwaarde voor de snelheid) is dan berekenbaar als de norm van de velocity $||vel(p)||$. De *acceleration* (versnellingsvector) is bijgevolg dan ook uit te drukken als de tweede afgeleide van $\mathbf{p}$: $acc(\mathbf{p}) = (vel(\mathbf{p}))'$. De afstand tussen twee moving points $\mathbf{p}$ en $\mathbf{q}$ bedraagt $||\mathbf{p} - \mathbf{q}||$ en de afstand tussen een moving point $\mathbf{p}$ en een gebied R bedraagt $min_{x \in R}\{||\mathbf{p} - x||\}$. De bewegingsvector van een moving point $\mathbf{p}$ is gedefinieerd als $dir(\mathbf{p}) = \frac{vel(\mathbf{p})}{||vel(\mathbf{p})||}$. Twee moving points $\mathbf{p}$ en $\mathbf{q}$ tenslotte bewegen in dezelfde richting als hun genormaliseerde bewegingsvectoren gelijk zijn.

Vervolgens kunnen enkele relaties gedefinieerd worden door gebruik te maken van de afstand (waarbij $dist(\mathbf{p}, \mathbf{q}, t) = ||\mathbf{p} - \mathbf{q}||$).

- $distance\_lessthan(\mathbf{p}, \mathbf{q}, t, d) \equiv (dist(\mathbf{p}, \mathbf{q}, t) \leq d)$.

- $in(\mathbf{p}, r, t) \equiv (\mathbf{p}(t)$ inside $r)$, waarbij r een gebied is en "inside" een spatial predikaat.
- $collision(\mathbf{p}, \mathbf{q}, t) \equiv (\mathbf{p}(t) = \mathbf{q}(t))$.
- $catching\_up(\mathbf{p}, \mathbf{q}, t_1, t_2) \equiv ((\forall t)(\forall t')(t_1 < t < t' < t_2) \Rightarrow dist(\mathbf{p}, \mathbf{q}, t) > dist(\mathbf{p}, \mathbf{q}, t'))$.

Gebruikmakend van de velocity kunnen volgende relaties uitgedrukt worden aan de hand van de richting (dir):

- $opposite\_direction(\mathbf{p}, \mathbf{q}, t) \equiv (dir(\mathbf{p})(t) + dir(\mathbf{q})(t) = 0)$.
- $same\_direction(\mathbf{p}, \mathbf{q}, t) \equiv (dir(\mathbf{p})(t) = dir(\mathbf{q})(t))$.
- $on\_collision\_course(\mathbf{p}, \mathbf{q}, t_1, t_2) \equiv ((\forall t)(t_1 < t < t_2 \Rightarrow opposite\_direction(\mathbf{p}, \mathbf{q}, t)) \wedge (\exists t)(t_2 < t \wedge collision(\mathbf{p}, \mathbf{q}, t)))$.
- $aim\_at(\mathbf{p}, r, t) \equiv ((\exists \mathbf{x})(\mathbf{x}\ inside\ r \wedge (unit(\mathbf{x} - \mathbf{p}(t)) = dir(\mathbf{p})(t))))$, waarbij $unit(\mathbf{p})$ de eenheidsvector van $\mathbf{p}$ is.
- $enter(\mathbf{p}, r, t) \equiv (on\_line(\mathbf{p}, boundary(r), t) \wedge (\exists t')(t' < t \wedge (\forall t'')(t' < t'' < t \Rightarrow \neg(\mathbf{p}(t)\ inside\ r))))$ waarbij "boundary" de grenslijnen van het gebied r bevat. De formule drukt uit dat $\mathbf{p}$ altijd buiten r is voordat het de omtrek van het gebied snijdt.

Het nut van deze relaties zal nog duidelijk blijken bij het opstellen van queries over het linear constraint model.

2.3.2 Het datamodel

Gebaseerd op de vorige sectie kan nu het effectieve datamodel gedefinieerd worden. Er worden lineaire constraints gebruikt om de functies van *time* naar $points_n$ te modelleren. Omwille van het feit dat tijd en plaats uitgedrukt worden in reële getallen moeten de constraints ook uitgedrukt worden in een tijdsvariabele t en n variabelen $x_1, x_2, \ldots, x_n$.

Definitie 2.2. Een moving point $\mathbf{p}$ heeft een *linear constraint represenation* als er een positief getal m bestaat en reële getallen $a_1, a_2, \ldots, a_m$ die voldoen aan:

- $a_1 < a_2 < \ldots < a_m$, en

- zij $a_0 = -\infty$ en $a_{m+1} = +\infty$ en voor elke $1 \leq i \leq n$ dan wordt de functie $p_i(t)$ voorgesteld als:

$$p_i(t) = \bigvee_{j=0}^{m} (x_i = b_{ij}t + c_{ij} \wedge a_j \leq t \leq a_{j+1}) \tag{2.1}$$

  waarbij b_{ij}'s en c_{ij}'s reële getallen zijn en voor elke $1 \leq j \leq m$ en elke $1 \leq i \leq n$ geldt:

$$b_{ij-1}a_j + c_{ij-1} = b_{ij}a_j + c_{ij} \tag{2.2}$$

□

Een moving point zoals in bovenstaande definitie kan op elke tijdsinstantie $a_1, a_2, \ldots, a_m$ van richting en snelheid veranderen. Elke x_i uit elke p_i van de definitie is continu en oneindig afleidbaar. Elke x_i is immers lineair in t dus is de eerste afgeleide constant (kan ook 0 zijn) en de tweede, derde, ... afgeleide steeds 0. Formule 2.2 drukt de continuïteit op de verschillende a_i's uit. Hier wordt formeel uitgedrukt dat op het eindpunt van een bepaald interval de locatie hetzelfde moet zijn als op het beginpunt van het daaropvolgende interval zoals reeds gezien in voorbeeld 2.2.

Voorbeeld 2.3. Het vliegtuig uit voorbeeld 2.2 voorgesteld in het linear constraint model ziet er als volgt uit:

$$x_1 = 2t - 40 \wedge 0 \leq t \leq 21 \vee x_1 = 2 \wedge 21 \leq t \leq 22 \vee x_1 = \tfrac{1}{2}t - 9 \wedge 22 \leq t \leq 47$$
$$x_2 = -t + 23 \wedge 0 \leq t \leq 21 \vee x_2 = -t + 23 \wedge 21 \leq t \leq 22 \vee x_2 = 1 \wedge 22 \leq t \leq 47$$
$$x_3 = 30 \wedge 0 \leq t \leq 21 \vee x_3 = -5t + 135 \wedge 21 \leq t \leq 22 \vee x_3 = -t + 47 \wedge 22 \leq t \leq 47$$

Merk op dat het traject maar beschreven is tot op het moment van de landing. De velocity's van het vliegtuig zijn $(2, -1, 0)$ van tijd 0 tot 21, $(0, -1, -5)$ van tijd 21 tot 22, en $(\frac{1}{2}, 0, -1)$ in het laatste interval. De respectievelijke snelheden zijn $\sqrt{5}$,$\sqrt{26}$ en $\frac{\sqrt{5}}{2}$. □

Het effectieve datamodel gebruikt moving points in linear representation zoals gedefinieerd in definitie 2.2 als primitieve types. Aan de hand van dit primitieve type kan een *Logical Data Type* (LDT) gedefinieerd worden. Een LDT is vergelijkbaar met een abstract data type (ADT), maar het verschil is dat een LDT geen abstractie is maar dat het ook toelaat spatial relations uit te drukken.

Definitie 2.3. Zij $\tau : \mathit{time} \rightarrow \mathit{real}$ een functie van tijd naar de reële getallen en n een positief getal dan is het *n-aire logische vector* type (LV type), genoteerd als $\mathbf{P}_n$, het type $\tau^n = \tau \times \cdots \times \tau$. Het domein van $\mathbf{P}_n$ is de

verzameling van alle moving points die een linear constraint representation hebben.

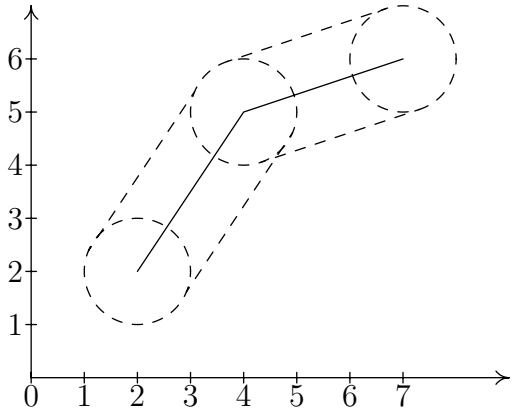
□

Een database in het linear constraint model heeft een *relatie-schema* dat bestaat uit een eindige verzameling (A, T) paren, waarbij A een attribuutnaam is en T een gewoon type of een LV-type, zodanig dat de attribuutnamen verschillend zijn. Een *tupel* over een relatie-schema R is een mapping van attribuutnamen in R op elementen uit het domein van het respectievelijke type. Een *relatie-instantie* over een schema R is een eindige verameling tupels over R. Een *database-schema* is een eindige set D van relatie-schema's en een *database-instantie* over D is een totale mapping I zodat voor elke R in D, $I(R)$ een relatie-instantie van R is.

2.4 Uncertainty in het trajectory location management datamodel

De datamodellen uit secties 2.1 - 2.3 zijn goede modellen voor het modelleren van moving points maar uiteraard moet er ook rekening gehouden worden met een zeker afwijking. Theoretisch gezien is het in de reeds besproken modellen mogelijk om de *exacte* locatie te berekenen van een moving object, maar dit is in de praktijk echter niet haalbaar. Er moet altijd rekening gehouden worden met een onzekerheid. In het geval dat de moving points auto's voorstellen kan het bijvoorbeeld zijn dat de auto's even moet afremmen voor een fietser, wat ervoor zal zorgen dat de auto zich vanaf dan niet meer exact op zijn veronderstelde locatie bevindt. Het is ook makkelijk in te zien dat het praktisch onmogelijk is om constant dezelfde snelheid aan te houden op een lijnsegment of binnen een tijdsinterval. In de volgende secties wordt het omgaan met uncertainty van het trajectory location management model besproken. In het point location management is het aspect uncertainty niet van toepassing.

In het trajectory location management systeem wordt uncertainty opgevangen door het gebruik van een uncertainty threshold u. Deze u geeft weer hoeveel de effectieve locatie van een object mag afwijken van de theoretische locatie. Dit betekent dat op elk punt van het traject er een cirkelvormig gebied rond dit punt is waarin het object zich effectief bevindt. Het idee achter deze uncertainty is dat er alleen updates plaatsvinden wanneer een moving object zich niet meer binnen de verwachte uncertainty threshold van de theoretische locatie bevindt. Een paar (Tr, u), met Tr een traject en u een geheel getal dat de uncertainty threshold is, wordt een *uncertainty trajectory*



Figuur 2.2: Traject en uncertainty van o

genoemd.

Voorbeeld 2.4. Zij o een moving point uit een MOD dat beweegt van punt $(2, 2)$ naar $(4, 5)$ en van $(4, 5)$ naar $(7, 6)$. Het traject is weergegeven in figuur 2.2 als volle lijn. Het eigenlijk gebied waarin o zich kan bevinden is getekend in stippellijn waarbij een uncertainty threshold van waarde 1 gehanteerd is. □

Definitie 2.4. Zij $T \equiv (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)$ een traject en u de uncertainty threshold, dan is op elk punt (x_i, y_i, t_i) de *r-uncertainty area* (of kortweg *uncertainty area*) de cirkel met straal u en middelpunt (x_i, y_i) waarbij (x_i, y_i) de verwachte locatie op tijdstip t_i is.

□

Merk op dat (x_i, y_i, t_i) niet noodzakelijk één van de punten (x_1, y_1, t_1), $(x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)$ is.

Definitie 2.5. Zij $T \equiv (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)$ een traject en u een uncertainty threshold, dan is een *Possible Motion Curve* PMC_r^T een continue functie $f_{PMC^T r} : time \rightarrow \mathbb{R}^2$ over het interval $[t_1, t_n]$ zodanig dat voor elke $t \in [t_1, t_n]$, $f_{PMC_r^T}(t)$ in de uncertainty area van de verwachte locatie op tijdstip t ligt.

□

De projectie van een possible motion curve op het XY-vlak wordt een *possible route* genoemd.

Definitie 2.6. Gegeven een uncertain trajectory (Tr, u) en twee punten $(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}) \in T_r$. Het *trajectory volume tussen* t_i en t_{i+1} van

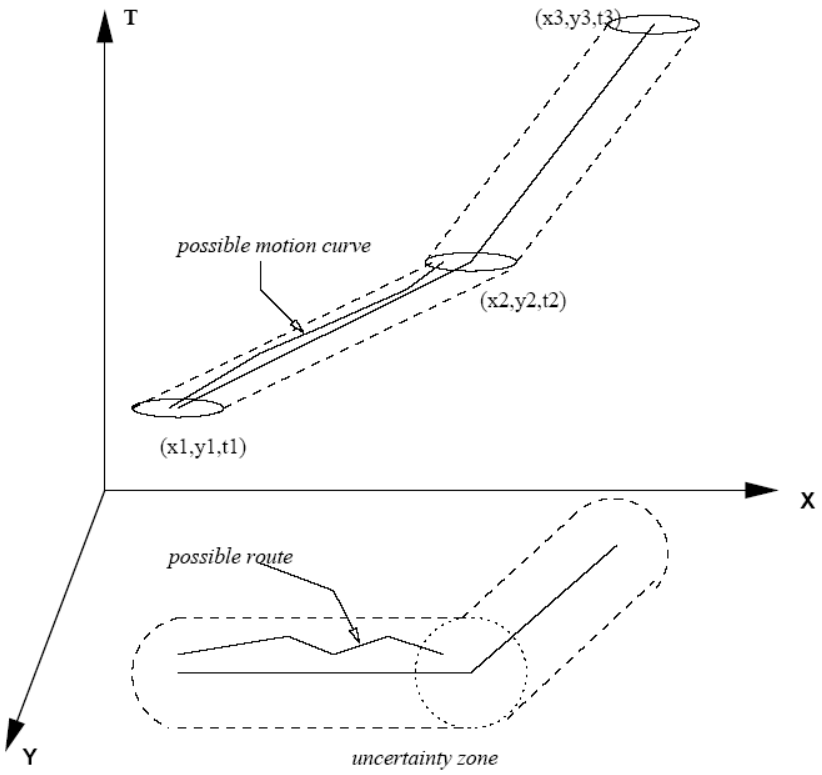
Tr is de verzameling van alle punten (x, y, t) die tot een PMC van Tr behoren met $t_i \leq t \leq t_{i+1}$. De tweedimensionale projectie wordt de *uncertainty zone* genoemd.

□

Definitie 2.7. Zij $T \equiv (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)$ een traject en u een uncertainty threshold, dan is het *trajectory volume* van Tr de verzameling van alle trajectory volumes tussen t_i en t_{i+1} $(i = 1, 2, \ldots, (n-1))$.

□

Merk op dat deze definities gebaseerd zijn op moving points die in een tweedimensionale ruimte bewegen. De gevallen voor andere dimensies zijn volledig analoog. De verschillende begrippen worden weergegeven in figuur 2.3.



Figuur 2.3: Possible Motion Curve en trajectory volume

Een vraag die zich echter wel stelt is: hoe wordt deze u gekozen? Wanneer voor u een grote waarde gekozen wordt, zal het gebied waarin het moving

point zich mogelijk bevindt groot worden. Als er voor u een kleine waarde gekozen wordt, zullen er veel updates nodig zijn wat veel resources verspilt. Een goede u is vooral afhankelijk van de applicatie: is deze gericht op een zeer laag verbruik van resources of een hoge precisie? In [TWZC02] wordt een kostenfunctie bestaande uit een afwijkingskost, communicatiekost en uncertainty kost geminimaliseerd om zo aan een optimale u te komen.

Hoofdstuk 3

Queries

In het vorige hoofdstuk werden twee belangrijke datamodellen besproken, maar uiteraard zijn niet alleen de datamodellen zelf belangrijk maar zijn ook de queries die erop gesteld kunnen worden van groot belang. In dit hoofdstuk worden de queries op het trajectory location management en het linear constraint datamodel voorgesteld. Het spreekt voor zich dat queries op het point location management datamodel volkomen nutteloos zijn.

3.1 Queries op het Trajectory Location Management datamodel

In het trajectory location management datamodel zit in wezen meer informatie dan hetgeen effectief in de database zit. Deze informatie moet wel uit de database gehaald kunnen worden. Wanneer er bvb. een query wordt gesteld die vraagt waar een object o zich bevindt op een tijdstip t, wat niet rechtstreeks in de database te vinden is, dan moet er geëxtrapoleerd worden. In de volgende secties worden de verschillende soorten queries en de uitvoering hiervan besproken.

3.1.1 Soorten queries

In het vorige hoofdstuk werden reeds enkele operatoren gedefinieerd op dit datamodel. Deze operatoren zijn ook queries op het het trajectory location management datamodel. De queries die gesteld kunnen worden kunnen in twee groepen opgedeeld worden: *point queries* en *range queries*. De twee soorten worden apart besproken.

Point Queries

Point queries zijn, zoals de naam reeds laat vermoeden, queries die betrekking hebben op één enkel moving point. De twee operatoren voor point queries zijn:

- $Where\_At(trajectory\ Tr, time\ t)$: Deze operator berekent de (theoretische) locatie van het object dat traject Tr volgt op tijdstip t.
- $When\_At(trajectory\ Tr, location\ l)$: Deze operator geeft een lijst van tijdstippen terug waarop het object dat traject Tr volgt op de locatie l passeert. Wanneer l niet op het traject Tr ligt, zal de verzameling punten C op Tr bepaald worden die het dichtst bij l liggen. Het resultaat is dan de verzameling van tijdstippen waarop het moving point een van de punten van C passeert.

De operatoren zijn relatief efficiënt uitvoerbaar. De $Where\_At$ operator kan uitgevoerd worden door binair te zoeken in $O(log(n))$ waarbij n het aantal lijnsegmenten van het traject is. De $When\_At$ operator moet alle lijnsegmenten sequentieel aflopen en kan dus uitgevoerd worden in $O(n)$. In de niet onredelijke veronderstelling dat een traject maximum uit een duizendtal lijnsegmenten bestaat kan dit dus in main memory uitgevoerd worden en zijn de complexiteiten aanvaardbaar.

Range queries

De range queries op het trajectory location management vormen een verzameling van booleaanse predikaten. De predikaten drukken altijd uit of een moving point in een gebied R komt in een tijdsinterval $[t_1, t_2]$. Door het gebruik van het model met uncertainty is dit niet eenduidig uitdrukbaar in één predikaat maar onstaat er een verzameling van predikaten. Dit komt omdat er drie verschillende aspecten in rekening gebracht moeten worden. Er kan gesteld worden of een conditie geldt

- ooit of altijd in het tijdsinterval $[t_1, t_2]$ ($sometime \leftrightarrow always$)
- ergens of overal in het tijdsinterval $[t_1, t_2]$ ($somewhere \leftrightarrow everywhere$)
- mogelijk of zeker in het tijdsinterval $[t_1, t_2]$ ($possibly \leftrightarrow definitely$)

Op elk van deze aspecten kan er dus het ene of het andere gesteld worden wat predikaten geeft zoals bvb. $Possibly\_Sometime\_Somewhere\_Inside$. Het aantal mogelijke predikaten dat gevormd kan worden is $2^3 \times 3! = 48$,

waarbij de volgorde ook van belang is (vandaar de vermenigvuldiging met 3!). Het is in dit geval echter wel zinloos om *everywhere* in een gebied uit te drukken met als gevolg dat er eigenlijk slechts $2^2 \times 2! = 8$ verschillende predikaten zijn. Een gebied wordt hier beschouwd als zijnde een gewone polygoon zonder gaten. PMC^T is een possible motion curve voor een gegeven uncertainty trajectory $T = (Tr, u)$ en t_1 en t_2 zijn tijdstippen. Alle mogelijk predikaten worden achtereenvolgens overlopen en van elk wordt een voorbeeld getoond in figuur 3.1.

- $Possibly\_Sometime\_Inside(T, R, t_1, t_2)$ is *true* als en slechts als er een PMC^T bestaat en een tijdstip t in het interval $[t_1, t_2]$ zodat PMC^T op tijdstip t door R loopt. M.a.w als de doorsnede van de uncertainty zone van T (gedurende het interval $[t_1, t_2]$) en het gebied R niet leeg is.

- $Sometime\_Possibly\_Inside(T, R, t_1, t_2)$ is *true* als en slechts als er een tijdstip t bestaat in het interval $[t_1, t_2]$ en een PMC^T zodat PMC^T op tijdstip t door het gebied R loopt. Merk op dat dit semantisch gezien hetzelfde is als het vorige, wat niet altijd zo zal zijn! Dit is omdat *Sometime* en *Possibly* als het ware existentiële kwantoren zijn, die dus kunnen omgewisseld worden, maar dit is niet bij alle predikaten het geval en zal dus ook niet altijd hetzelfde zijn als het omgekeerde.

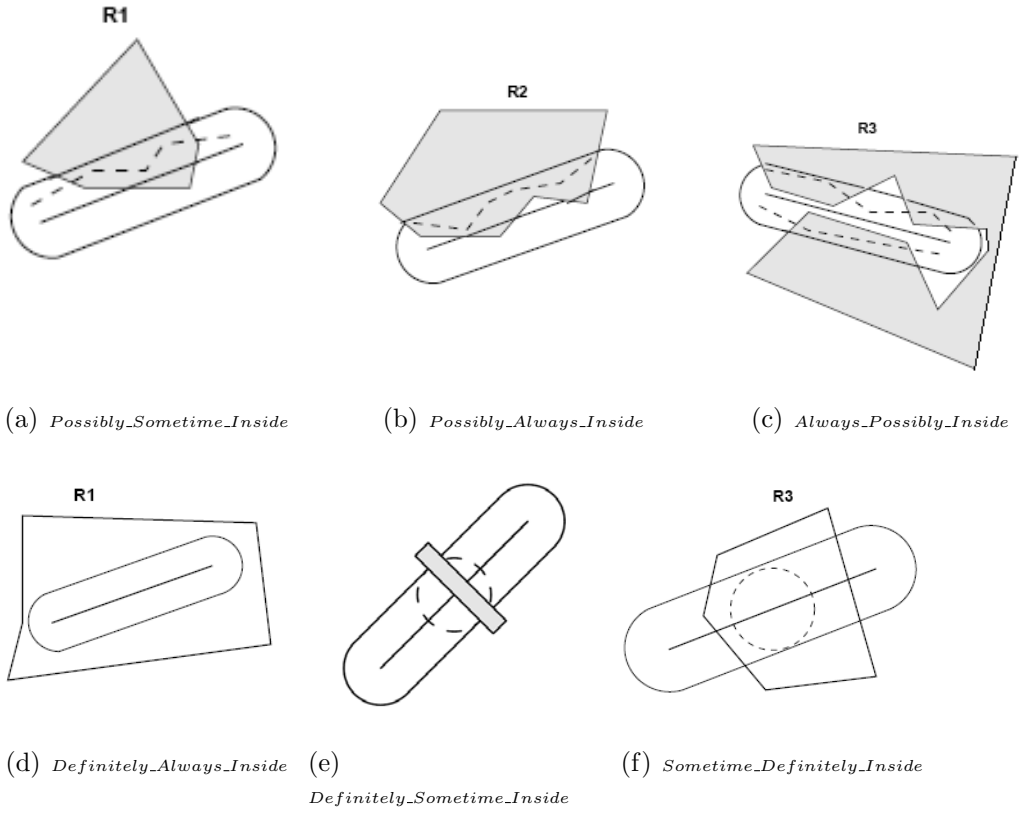
- $Possibly\_Always\_Inside(T, R, t_1, t_2)$ is *true* als en slechts als er een PMC^T bestaat die op elk tijdstip t in het interval $[t_1, t_2]$ door het gebied R loopt. M.a.w bestaat er een traject in de uncertainty zone van T dat gedurende het hele tijdsinterval in R ligt.

- $Always\_Possibly\_Inside(T, R, t_1, t_2)$ is *true* als en slechts als er voor elk tijdstip t in het tijdsinterval $[t_1, t_2]$ een PMC^T bestaat die op dat tijdstip door R loopt. Dit predikaat is semantisch verschillend van het vorige, dit is duidelijk te zien in figuur 3.1.

- $Always\_Definitely\_Inside(T, R, t_1, t_2)$ is *true* als en slechts als voor elke t in het tijdsinterval $[t_1, t_2]$ elke PMC^T op tijdstip t door het gebied R loopt.

- $Definitely\_Always\_Inside(T, R, t_1, t_2)$ is *true* als en slechts als voor elke PMC^T geldt dat deze op elke t in het tijdsinterval $[t_1, t_2]$ door het gebied R loopt. Merk op dat dit predikaat opnieuw semantisch equivalent is met het vorige.

- $Definitely\_Sometime\_Inside(T, R, t_1, t_2)$ is *true* als en slechts als voor elke PMC^T er een tijdstip t bestaat in het tijdsinterval $[t_1, t_2]$ zodat PMC^T op tijdstip t door het gebied R loopt.



(a) *Possibly_Sometime_Inside* (b) *Possibly_Always_Inside* (c) *Always_Possibly_Inside*

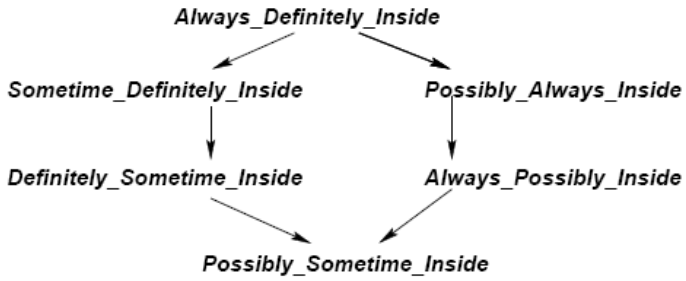
(d) *Definitely_Always_Inside* (e) *Definitely_Sometime_Inside* (f) *Sometime_Definitely_Inside*

Figuur 3.1: De zes predikaten voor range queries op het trajectory location management systeem

- $Sometime\_Definitely\_Inside(T, R, t_1, t_2)$ is *true* als er een tijdstip t bestaat in het interval $[t_1, t_2]$ waarop elke PMC^T door het gebied R loopt. Dit predikaat is semantisch verschillend met het vorige wat ook getoond wordt in de voorbeeldjes in figuur 3.1.

Eigenschap 3.1. Zij $T = (Tr, u)$ een uncertainty trajectory, R een polygoon en t_1 en t_2 twee tijdstippen dan geldt dat als $Possibly\_Always\_Inside(T, R, t_1, t_2)$ voldaan is, ook $Always\_Possibly\_Inside(T, R, t_1, t_2)$ voldaan is.

Het omgekeerde van de stelling is echter niet waar. Dit kan ingezien worden in figuur 3.1 waar het predikaat $Always\_Possibly\_Inside$ voldaan is door twee verschillende PMC^T's en wat dus ook niet kan voldoen aan $Possibly\_Always\_Inside$ omdat er daar één PMC^T moet bestaan die voldoet voor elke t. Mits een extra voorwaarde is het omgekeerde van voorgaande stelling echter wel voldaan:



Figuur 3.2: Relaties tussen de verschillende predikaten

Eigenschap 3.2. Zij $T = (Tr, u)$ een uncertainty trajectory, R een convexe polygoon en t_1 en t_2 twee tijdstippen dan geldt $Possibly\_Always\_Inside(T, R, t_1, t_2)$ als en slechts als $Always\_Possibly\_Inside(T, R, t_1, t_2)$.

Door het stellen van de extra conditie dat de polygoon convex moet zijn, is het niet meer mogelijk om een geval te hebben zoals in figuur 3.1(c). Bij de predikaten $Definitely\_Sometime\_Inside$ en $Sometime\_Definitely\_Inside$ kan een gelijkaardige, maar niet dezelfde, vaststelling gedaan worden:

Eigenschap 3.3. Zij $T = (Tr, u)$ een uncertainty trajectory, R een polygoon en t_1 en t_2 twee tijdstippen dan geldt dat als $Sometime\_Definitely\_Inside$ (T, R, t_1, t_2) voldaan is, ook $Definitely\_Sometime\_Inside(T, R, t_1, t_2)$ voldaan is.

Het omgekeerde van voorgaande stelling is echter niet waar, ook niet wanneer R convex zou zijn. Dit kan ingezien worden in het voorbeeld in figuur 3.1(e) waarbij $Definitely\_Sometime\_Inside$ voldaan is en de polygoon convex is maar $Sometime\_Definitely\_Inside$ niet voldaan is. Er bestaat geen punt op het traject waarvoor het hele uncertainty-gebied, cirkel met straal u waar het moving point zich zou kunnen bevinden, volledig in de polygoon ligt.

De bewijzen van de twee stellingen kunnen geconstrueerd worden aan de hand van de eigenschap: $(\exists x)(\forall y)P(x, y) \Rightarrow (\forall y)(\exists x)P(x, y)$. Puur semantisch bekeken zijn er dan in principe nog maar 6 verschillende predikaten. De onderlinge relatie tussen de predikaten is weergegeven in 3.2 ([TWZC02]).

Een voorbeeld van een mogelijke query die nu gesteld kan worden door gebruik te maken van de gedefinieerde operatoren en predikaten is: *"geef alle objecten die mogelijk altijd in een gebied R zijn tussen het tijdstip waarop het*

object A op locatie L_1 is en het tijdstip waarop A op locatie L_2 is" als de query:

$$Possibly\_Always\_Inside(T, R, When\_At(T_A, L_1), When\_At(T_A, L_2))$$

3.2 Queries op het Linear Constraint Model

Net zoals bij het trajectory location management datamodel moet er ook bij het linear constraint model aandacht besteed worden aan het stellen van queries over het model. In de volgende sectie wordt de query-taal voorgesteld door Su, Xu en Ibarra [SXI01] besproken.

3.2.1 Query-taal

Voor het gebruik van het linear constraint model wordt de relationele calculus uitgebreid tot een constraint query-taal. Een eerste-orde formule $\varphi(\overline{x})$, met $\overline{x}$ de vrije variabelen uit de formule, is een query over het linear constraint model zodat: zij I een constraint database, het resultaat van query Q op I gedefinieerd is als:

$$Q(I) = \{\overline{a} |\ de\ database\ I \models \varphi(\overline{a})\}$$

Een belangrijke opmerking hierbij is dat het resultaat van een query een oneindige relatie kan zijn terwijl het definieerbaar zou moeten zijn als een constraint relatie. Dit probleem kan verholpen worden door kwantor-eliminatie. Door het elimineren van de kwantoren uit de eerste-orde formule die de query voorstelt, wordt er een constraint relatie bekomen. Kwantor-eliminatie wordt verderop besproken in sectie 3.2.2.

De relationele calculus kan uitgebreid worden voor moving objects. Hiervoor moeten een aantal extra zaken toegelaten worden. Er worden reële variabelen (om ruimtelijke coördinaten en tijd voor te stellen) bijgevoegd om de data uit de LV-types uit te drukken, lineaire constraints over deze reële variabelen en eerste-orde constructies ($\wedge, \vee, \neg, \forall, \exists$) op de constraints.

Definitie 3.1. Een *term* in de uitgebreide calculus bestaat uit:

- Variabelen van een gewoon type of van het LV-type.
- Waardes uit het domein van gewone types of van het LV-type.
- Optelling van twee reële (of tijds-) termen.
- Vermenigvuldiging van een reële (of tijds-) term met een reëel getal.

of één van de volgende gevallen:

- Zij $\mathbf{p}$ en $\mathbf{q}$ twee termen van het type $\mathbf{P}_n$ en zij c een reële term, dan zijn ook $\mathbf{p} + \mathbf{q}$ en $c.\mathbf{p}$ termen van het type $\mathbf{P}_n$.
- Zij $\mathbf{x}$ een vector, dan is ook $unit(\mathbf{x})$ een term.
- Zij $\mathbf{p}$ een term van het type $\mathbf{P}_n$, dan zijn $vel(\mathbf{p})$ en $dir(\mathbf{p})$ ook termen van het type $\mathbf{P}_n$ en is $len(\mathbf{p})$ een reële term.

□

Vermits een n-aire vector van reële termen ook gezien kan worden als een waarde van het type $\mathbf{P}_n$ (met elke component een constante functie) worden de operaties op LV-types uitgebried om ook op vectoren van reële termen van toepassing te zijn. Zij $\mathbf{p}$ een element uit het domein $\mathbf{P}_n$, zij a een tijdstip en zij $b_1, b_2, \ldots, b_n$ n reële getallen, dan is de expressie $\mathbf{p}(a; b_1, b_2, \ldots, b_n)$ waar als het moving point $\mathbf{p}$ op tijdstip a zich op positie $(b_1, b_2, \ldots, b_n)$ bevindt.

Definitie 3.2. De *formules* in de uitgebreide calculus bestaan uit:

- $x = y$ als x en y termen zijn van hetzelfde type, of $x \leq y$ indien x en y reële of tijdstermen zijn.
- $\mathbf{p}(t; x_1, x_2, \ldots, x_n)$ met $\mathbf{p}$ een term van het type $\mathbf{P}_n$, t een tijdsterm en $x_1, x_2, \ldots, x_n$ reële termen.
- $R(x_1, x_2, \ldots, x_n)$ met R een relatieschema in de database en de x_i's termen van de respectievelijke types.
- $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \exists x\varphi$ en $\forall x\varphi$ met φ en ψ formules en x een variabele.

□

Uiteindelijk kan dan een query gedefinieerd worden als volgt:

Definitie 3.3. Een *query* is een expressie van de vorm $\{(x_1, x_2, \ldots, x_n) \mid \varphi\}$ waarvan alle vrije variabelen uit φ in $\{x_1, x_2, \ldots, x_n\}$ zitten. □

Voorbeeld 3.1. Beschouw het volgende database-schema:

FLIGHTS($airline : string, number : string, position : moving\_point_3$)
AIRPORTS($code : string, country : string, location : region_2$)
RUNWAYS($airport : string, number : string, landing\_direction : real \times real$)

CITIES($name : string, country : string, area : region_2$)

en enkele queries:

q_1: Lokaliseer alle Ryanair vluchten boven Brussel op 29 januari 2000 om 16h.

q_2: Geef alle paren van vluchten binnen 300km van de luchthaven van Zaventem op dit moment die in tegengestelde richting vliegen.

q_3: Geef alle vluchten die de laatste 15 minuten boven Parijs vlogen.

De queries kunnen in de geconstrueerde query-taal uitgedrukt worden als volgt (de relatienamen worden afgekort tot de eerste twee letters):

1. Query q_1 kan uitgedrukt worden als:

$$\{(n, x_1, x_2, x_3) | (\exists \mathbf{p})(\exists r)(\text{FL}(\text{Ryanair}, n, \mathbf{p}) \wedge \mathbf{p}(t_0; x_1, x_2, x_3) \wedge \text{CI}(\text{Brussel}, \text{BE}, r) \wedge r(x_1, x_2))\}$$

   waarbij t_0 het tijdstip "29 januari 2000, 16h" voorsteld. r is een tweedimensionaal gebied en $r(x, y)$ geldt wanneer het punt (x, y) in het gebied r ligt.

2. Query q_2 kan, vermits alle vliegtuigen aan een constante snelheid vliegen, als volgt worden uitgedrukt:

$$\begin{aligned}\{(n_1, n_2) | &(\exists p)(\exists q)(\exists x_1)(\exists x_2)(\exists r)(\exists p)\text{FL}(x_1, n_1, \mathbf{p}) \wedge \text{FL}(x_2, n_2, \mathbf{q}) \\ &\wedge \text{AI}(\text{Za}, \text{BE}, r) \wedge dir((p))(t_{now}) + dir((q))(t_{now}) = 0 \wedge r(\mathbf{p}) \\ &\wedge (\exists y_1)(\exists y_2)(\exists y_3)(\exists \mathbf{p})(t_{now}; y_1; y_2, y_3) \wedge len(\mathbf{p} - (y_1, y_2)) \leq 300 \\ &\wedge (\exists z_1)(\exists z_2)(\exists z_3)(\exists \mathbf{q})(t_{now}; z_1, z_2, z_3) \wedge len(\mathbf{p} - (z_1, z_2)) \leq 300\}\end{aligned}$$

   waarbij t_{now} staat voor de huidige tijd. Het eerste gedeelte zoekt de locaties van twee vluchten en de luchthaven. Het tweede gedeelte zorgt ervoor dat de twee vluchten in tegengestelde richting vliegen en de twee laatste delen zorgen ervoor dat de afstand van de twee vluchten tot de luchthaven kleiner is dan 300km.

3. Query q_3 drukt uit dat een vliegtuig de laatste 15 minuten over Parijs vloog door uit te drukken dat het op elke tijdstip tussen nu en 15 minuten geleden boven Parijs vloog.

$$\begin{aligned}&\{(x, n) | (\exists \mathbf{p})(\exists r)\text{FL}(x, n, \mathbf{p}) \wedge \text{CI}(Paris, FR, r) \wedge \\ &(\forall t)(t_{now} - 15 \leq t \leq t_{now} \Rightarrow (\exists x_1)(\exists x_2)(\exists x_3)(\mathbf{p}(t; x_1, x_2, x_3) \wedge r(x_1, x_2)))\}\end{aligned}$$

□

3.2.2 Kwantor-eliminatie

In 1954 publiceerde Alfred Tarski zijn bewijs dat $(\mathbb{R}, +, \times, 0, 1, <)$ kwantor-eliminatie toelaat. Dit houdt in dat er een algoritme bestaat dat voor elke formule $\varphi(x_1, x_2, \ldots, x_n)$ over $(+, \times, 0, 1, <)$ een kwantorvrije formule $\psi(x_1, x_2, \ldots, x_n)$ geeft, die equivalent is met $\varphi(x_1, x_2, \ldots, x_n)$. Een welbekend voorbeeld is het volgende:

Voorbeeld 3.2. Beschouw de formule $\varphi(a, b, c) =$

$$a \neq 0 \wedge (\exists x)(ax^2 + bx + c = 0).$$

Door een eenvoudige wiskundige afleiding kan de equivalente kwantorvrije formule $\psi(a, b, c) =$

$$a \neq 0 \wedge (b^2 - 4ac \geq 0)$$

gevonden worden. □

Een volledige beschrijving van het algoritme van Tarski zou hier echter te ver leiden.

Hoofdstuk 4

Neighborhood-begrippen voor spatial data mining

Alvorens verder te gaan met spatial data mining, zijn er eerst enkele begrippen nodig die in het volgende hoofdstuk gebruikt gaan worden. Deze begrippen worden in dit hoofdstuk besproken. De betreffende begrippen zijn de neighborhood-relaties, de neighborhood-grafen en hun operaties en neighborhood-indices. Elk van deze drie worden in een aparte sectie besproken.

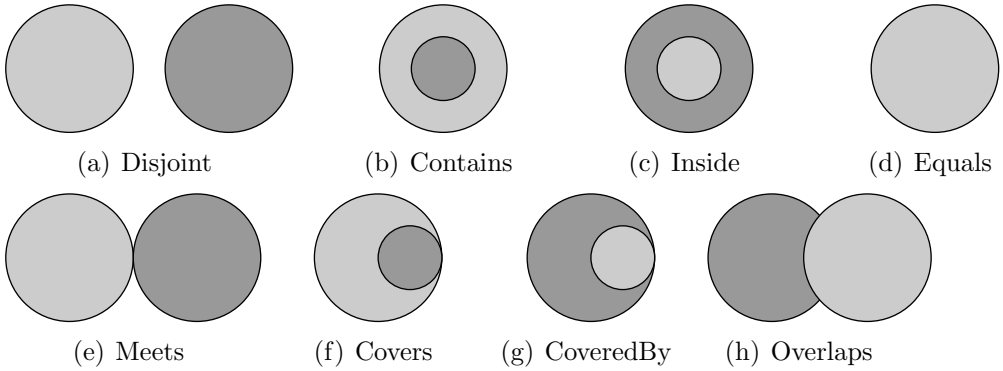
4.1 Neighborhood-relaties

Een eerste zeer belangrijk begrip is *neighborhood-relaties*. Wanneer er naar patronen gezocht wordt, is het niet enkel interessant om allerlei informatie te weten over bepaalde objecten, maar spelen de nabijgelegen objecten ook een zeer belangrijke rol want zij hebben dikwijls een niet onbelangrijke invloed op de beschouwde objecten. Een eenvoudig voorbeeld hiervan is een fabriek die zich ergens vestigt. De luchtvervuiling op verschillende locaties rond de fabriek wordt dan sterk beïnvloedt door het bouwen van de fabriek. Er zijn drie verschillende neighborhood-relaties die afzonderlijk besproken zullen worden: *topologische relaties*, *afstandsrelaties* en *richtingsrelaties*.

Spatial objecten kunnen punten, rechtes, veelhoeken of veelvlakken zijn, voorgesteld door één of meerdere punten.

4.1.1 Topologische relaties

Topologische relaties zijn die relaties die invariant zijn onder de topologische transformaties. De topologische relatie tussen twee objecten blijft dus dezelf-



Figuur 4.1: 8 topologische relaties

de, ook wanneer de objecten gedraaid, verschoven, gescaleerd, ... worden. De definities van de topologische relaties zijn gebaseerd op de relaties tussen het inwendige en de rand van twee objecten.

Definitie 4.1. De topologische relaties tussen twee objecten A en B zijn afgeleid van de intersecties tussen de randen en de inwendigen van A en B. De verschillende relaties zijn: *A disjoint B*, *A contains B*, *A inside B*, *A equals B*, *A meets B*, *A covers B*, *A coveredBy B* en *A overlaps B*. □

De acht relaties worden weergegeven in figuur 4.1. De A's uit de definitie zijn de lichtgrijze objecten en de B's zijn de donkergrijze. Er zijn in theorie meerdere mogelijkheden voor relaties tussen rand en inwendige van een object, meerbepaald $2^4 = 16$ maar de relaties die er niet tussenzitten zijn onmogelijke relaties. Wanneer bijvoorbeeld de intersectie tussen het inwendige van beide objecten niet leeg is maar de intersecties tussen de rand van de ene en het inwendige van de andere wel, dan is het niet mogelijk dat de intersectie tussen de randen leeg is en drukt dit dus een niet realistische relatie uit.

4.1.2 Afstandsrelaties

Een volgende belangrijke klasse van neighborhood-relaties zijn de *afstandsrelaties*. De afstandsrelaties vergelijken de afstand tussen twee objecten met een constante. De afstand *dist* tussen twee objecten (een verzameling van punten) is dan simpelweg de kortste afstand tussen de twee verzamelingen van punten.

Definitie 4.2. Zij *dist* een afstandsfunctie, θ een vergelijkingsoperator $<, >$ of $=$, c een reëel getal en A en B spatial objecten ($A, B \in 2^{Points}$) dan geldt de afstandsrelatie $A\ distance_{\theta\ c}\ B$ als en slechts als $dist(A, B)\ \theta\ c$. □

Deze definitie bepaalt hier enkel een notatie die in dit en volgende hoofdstukken gebruikt zal worden.

4.1.3 Richtingsrelaties

De laatste klasse van neighborhood-relaties zijn de richtingsrelaties. Voor het definiëren van richtingsrelaties $B\ rel\ A$ tussen twee objecten wordt er een onderscheid gemaakt tussen het bronobject A en het doelobject B. Merk op dat het bronobject vanachter staat in de uitdrukking en het doelobject van voor. Voor het bronobject wordt er een representatief punt $rep(A)$ binnen het object bepaald, voor het doelobject worden alle punten behouden. Een representatief punt voor een object zou bijvoorbeeld het middelpunt kunnen zijn. Dit representatief punt wordt gebruikt om een assenstelsel te construeren waarvan de kwadranten dan de richting bepalen.

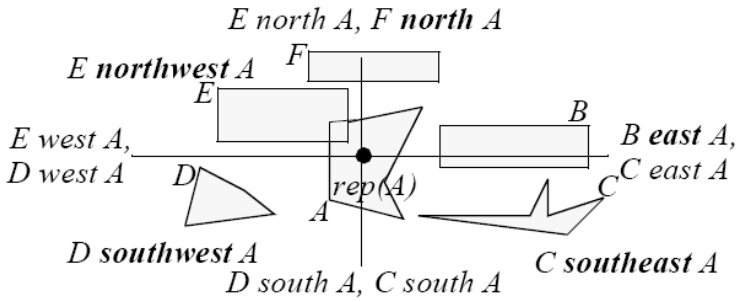
Definitie 4.3. Zij $rep(A)$ een representatief punt van het bronobject A dan geldt:

- B *noord-oost* A als en slechts als $\forall b \in B : b_x \geq rep(A)_x \wedge b_y \geq rep(A)_y$ (*zuid-oost*, *zuid-west* en *noord-west* zijn op een analoge manier gedefinieerd).
- B *noord* A als en slechts als $\forall b \in B : b_y \geq rep(A)_y$ (*zuid*, *west* en *oost* zijn analoog gedefinieerd).
- B *willekeurige_richting* A voor alle A, B.

□

De relaties uit het eerste puntje van de definitie gelden als het doelobject volledig in één kwadrant ligt. De relaties uit het tweede puntje gelden wanneer het doelobject in maximum twee aangrenzende kwadranten ligt (m.a.w volledig aan één zijde van één van de assen). Het derde puntje geldt uiteraard altijd. De verschillende relaties zijn weergegeven in figuur 4.2 ([EGKS99]). De relatie tussen twee objecten is niet eenduidig bepaald, wanneer een object *noord-oost* ligt van een bronobject ligt het automotisch ook *noord* en *oost* van dat object en uiteraard geldt ook nog *willekeurige_richting*. Om dit te verhelpen wordt de exacte richting bepaald, zijnde de meest specifieke relatie die geldt. De exacte richtingen in figuur 4.2 zijn in het vet aangeduid.

Definitie 4.4. Zij r_1 en r_2 twee richtingsrelaties dan is r_2 een *specialisatie* van r_1, genoteerd als $r_2\{r_1$ als en slechts als $\forall A \in 2^{Points}, B \in 2^{Points} : A\ r_2\ B \Rightarrow A\ r_1 B$. □



Figuur 4.2: Illustatratie van de richtingsrelaties

Definitie 4.5. Zij A en B spatial objecten ($A, B \in 2^{Points}$) dan is r de *exacte richtingsrelatie* van A en B als en slechts als $\forall r^* \in DirectionRelations$: $A\ r\ B \wedge A\ r^*\ B \Rightarrow r \{ r^*$. □

De drie verschillende neighborhood-relaties mogen gecombineerd worden aan de hand van de logische operatoren $\wedge$ en $\vee$. De samengestelde relaties worden *complexe neighborhood-relaties* genoemd.

Definitie 4.6. Zij r_1 en r_2 neighborhood-relaties dan zijn $r_1 \wedge r_2$ en $r_1 \vee r_2$ ook opnieuw neighborhood-relaties, *complexe neighborhood-relaties* genaamd. □

4.2 Neighborhood-grafen en hun operaties

In deze sectie worden neighborhood-grafen en hun operaties besproken. Eerst zullen de begrippen neighborhood-graaf en neighborhood-pad besproken worden en daarna zullen nog een aantal operaties hierop gedefinieerd worden.

4.2.1 Neighborhood-grafen en paden

Gebruikmakend van de neighborhood-relaties, besproken in de vorige sectie, kunnen er neighborhood-grafen geconstrueerd worden. Neighborhood-grafen en paden worden formeel gedefinieerd als volgt:

Definitie 4.7. Zij *neighbor* een neighborhood-relatie en $DB \subseteq 2^{Points}$ een database van objecten.

a) Een *neighborhood-graaf* $G^{DB}_{neighbor} = (N, E)$ is een graaf met als knopen de verzameling N van alle objecten $o \in DB$ en als bogen de verzameling $E \subseteq N \times N$ waarbij twee knopen n_1 en n_2 verbonden zijn als en slechts als $neighbor(n_1, n_2)$ geldt. Zij n het aantal knopen in de graaf en e het

aantal bogen dan is $f = e/n$ de *fan-out*. De fan-out is het gemiddeld aantal bogen per knoop.

b) Een *neighborhood-pad* is een sequentie van knopen $[n_1, n_2, \ldots, n_k]$ waarbij $neighbor(n_i, n_{i+1})$ geldt voor elke $1 \leq i \leq n-1$. De *lengte* van het pad is k.

□

Wanneer er voor een gegeven spatial database een neighborhood-graaf opgesteld wordt, zal deze al gauw heel veel paden kunnen voortbrengen. Een groot deel van de paden dat al geëlimineerd kan worden zijn de paden met lussen in, maar zonder deze zullen er nog steeds veel paden voortgebracht kunnen worden. In de praktijk gaat echter maar een relatief klein gedeelte van al deze paden interessant zijn. De paden die interessant zouden kunnen zijn, zijn enkel de paden die zich weg van een object bewegen. Het is dus zeer belangrijk dat, wanneer er gebruik gemaakt wordt van neighborhood-grafen, er eerst filters worden toegepast die het aantal paden aanzienlijk verminderen. Het concept 'filters' komt in de volgende secties nog aan bod.

4.2.2 Enkele operaties

Om te beginnen worden de operaties selectie, unie, doorsnede en verschil verondersteld aanwezig te zijn. Een eerste nieuwe operatie is de operatie $neighbors$:

$$neighbors : NGrafen \times Objecten \times Predikaten \rightarrow 2^{Objecten}$$

waarbij $NGrafen$ de verzameling van alle neighborhood-grafen voorstelt. De operatie $neighbors(g, o, pred)$ bepaalt de verzameling van objecten die verbonden zijn met het object o via een boog van graaf g en die aan predikaat $pred$ voldoen. Een volgende operatie die toegevoegd wordt is:

$$extensions : NGrafen \times 2^{NPaden} \times Integer \times Predikaten \rightarrow 2^{NPaden}$$

De operatie $extensions(g, paden, max, pred)$ geeft de verzameling van alle paden terug die een uitbreiding zijn van een van de paden in *paden* met maximum max knopen uit de graaf g en die aan het predikaat *pred* voldoen. De paden in *paden* zitten niet bevat in het resultaat, dit impliceert dat een leeg resultaat wijst op het feit dat er geen enkel pad uit *paden* uitgebreid kon worden. Omdat het resultaat van deze operatie al vlug heel erg groot kan worden, is het de meest cruciale operatie betreffende de efficiëntie. Het

predikaat *pred* geldt dan ook als filter om het aantal sterk te reduceren. Twee mogelijke filters zijn de filter *starlike* en *variable-starlike*.

Definitie 4.8. Zij $p = [n_1, n_2, \ldots, n_k]$ een neighborhood-pad en rel_i de exacte richting tussen n_i en n_{i+1} dan zijn de de predikaten *starlike* en *variable starlike* voor een pad p gedefinieerd als volgt:

$$starlike(p) \Leftrightarrow \begin{cases} (\exists j < k : \forall j < i < k : n_{i+1}\ rel_i\ n_i \Leftrightarrow rel_i\{rel_j) & k \geq 3 \\ true & k < 3. \end{cases}$$

$$variable\text{-}starlike(p) \Leftrightarrow \begin{cases} (\exists j < k : \forall j < i < k : n_{i+1}\ rel_i\ n_i \Leftrightarrow rel_i\{rel_1) & k \geq 3 \\ true & k < 3\ . \end{cases}$$

Merk op dat het verschil tussen beide in het laatste gedeelte zit: $rel_i\{rel_j$ bij *starlike* en $rel_i\{rel_1$ bij *variable-starlike*.

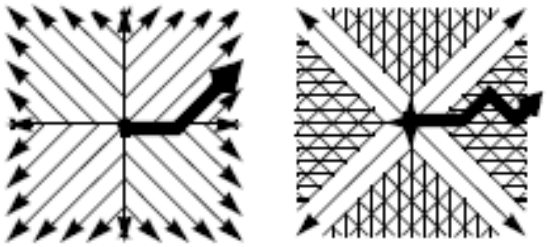
□

De definitie van *starlike* drukt uit dat een pad alleen uitgebreid kan worden met een knoop zodat de richting van het pad hetzelfde blijft of een specialisatie is van de vorige richting, maar dus nooit veralgemeent. Stel bijvoorbeeld dat de richting van een pad *noord* is, dan mag het pad enkel uitgebreid worden met een extra knoop wanneer de richting *noord* blijft of verandert naar *noord-oost* of *noord-west*. Dit is grafisch weergegeven in figuur 4.3(a) ([EGKS99]. Merk op dat wanneer een pad eens in één van de vier meest specifieke richtingen gaat dit ook zo moet blijven, wat duidelijk te zien is in figuur 4.3(a).

De definitie van *variable-starlike* is iets minder streng dan deze van *starlike*. De definitie van *variable-starlike* zegt dat een knoop enkel mag worden toegevoegd wanneer de richting hetzelfde of een specialisatie is van de initiële richting (de richting van de eerste naar de tweede knop) van het pad. Dit is minder beperkend dan *starlike* omdat een richting terug veralgemeend kan worden zolang het maar geen veralgemening is van de initiële richting. *variable-starlike* is grafisch voorgesteld in figuur 4.3(b). Bij *variable-starlike* is het enige geval waarop een richting niet meer kan veranderen wanneer de initiële richting reeds één van de meest specifieke richtingen was.

4.3 Neighborhood-indices

Een veel voorkomende indexeringsmethode, die dikwijls gebruikt wordt in spatial databases, is de R-tree. Indices zoals R-trees zijn zeer nuttig bij het



(a) *starlike* (b) *variable-starlike*

Figuur 4.3: De filters *starlike* en *variable-starlike*

gebruik van region searches en nearest neighbor queries. Wanneer de spatial objecten echter zeer complex zijn, zal het zoeken van de nabijgelegen buren een zeer dure taak blijven. Het zou dus een goede zaak zijn de relevante neighborhood-relaties bij te houden, zonder telkens de complexe objecten te moeten gebruiken. Om dit te realiseren is de neighborhood-index ingevoerd.

Alvorens verder te gaan met de neighborhood-index dient eerst nog volgende definitie gegeven te worden:

Definitie 4.9. Zij r een neighborhood-relatie dan is de *critical distance* van r ($c\text{-}distance(r)$) gedefinieerd als volgt:

$$c\text{-}distance(r) = \begin{cases} 0 & \text{indien } r \text{ een topologische relatie is behalve } disjoint \\ c & \text{indien } r \text{ de relatie } distance_{<c} \text{ of } distance_{=c} \text{ is} \\ \infty & \text{indien } r \text{ een richtingsrelatie is, de afstandsrelatie} \\ & distance_{>c} \text{ of de topologische relatie } disjoint \\ min(c\text{-}distance(r_1), c\text{-}distance(r_2)) & \text{indien } r = r_1 \wedge r_2 \\ max(c\text{-}distance(r_1), c\text{-}distance(r_2)) & \text{indien } r = r_1 \vee r_2 \end{cases}$$

□

Er kan nu verdergegaan worden met de definitie van een neighborhood-index.

Definitie 4.10. Zij DB een verzameling van spatial objecten, c en $dist$ reële getallen, D een richtingspredikaat en T een topologisch predikaat dan is de *neighborhood-index* voor DB met maximum afstand c, genoteerd als N_c^{DB}, gedefinieerd als volgt:

$$N_c^{DB} = \{(o_1, o_2, dist, D, T) | o_1\ distance_{=dist}\ o_2 \wedge dist \leq c \wedge o_2\ D\ o_1 \wedge o_1\ T\ o_2\}$$

□

neighbors(graaf G_r^{DB}, object o, predikaat *pred*)

- Index selectie: selecteer de kleinste beschikbare neighborhood index N toepasbaar op G_r^{DB}.
- Filter stap:
  - indien N bestaat, haal dan alle buren van o uit N
  - indien N niet bestaat, gebruik de spatial index en neem de verzameling objecten o' die voldoen aan $o\ r\ o'$
- Filtering: van de reeds bekomen objecten, geef de objecten o' terug die voldoen aan zowel $o\ r\ o'$ als $pred(o')$

Figuur 4.4: *neighbors*-algoritme

Definitie 4.11. Een neighborhood-index N_c^{DB} is *toepasbaar* op een neighborhood graaf G_r^{DB} als $c \geq c\text{-}distance(r)$.

□

Het algoritme om neighbors te bepalen met behulp van de kleinste beschikbare neighborhood index is gegeven in figuur 4.4. De belangrijkste factor van het algoritme om extensions te bepalen is het bepalen van de neighbors, maar uiteraard wordt daarvoor het efficiënte neighbor-algoritme gebruikt wat ervoor zorgt dat de extensions vlot bepaald kunnen worden. Het algoritme om de extensions te bepalen is weergegeven in figuur 4.5

extensions(graaf g, paden p, integer *max-length*, filter f)

```
1: initialiseer de lijst van kandidaat paden als de verzameling paden p
2: while er is een kandidaat pad met een lengte < max-length do
3:    haal het pad uit de verzameling en noem het kand;
4:    o := laatste knoop van kand;
5:    neighborhood = neighbors(g, o, TRUE);
6:    for elke neighbor in neighborhood do
7:       maak een uitbreiding ext van kand met neighbor
8:       if  ext is geldig AND ext voldoet aan de filter f then
9:          geef ext terug als een resultaat
10:         voeg ext vooraan toe aan de lijst van kandidaat paden
11:      end if
12:   end for
13: end while
```

Figuur 4.5: *extensions*-algoritme

Hoofdstuk 5

Spatial Mining

Eens er goed gedefineerde datamodellen beschikbaar zijn alsook queries hierop, kan er op de data gemined worden. In dit hoofdstuk worden een aantal spatial patronen gedefinieerd en algoritmes beschreven om naar deze patronen te minen. In de volgende secties worden eerst de verschillende spatial patronen besproken en daarna worden de algoritmes hiervoor voorgesteld.

5.1 Spatial Patronen

Er zijn een aantal verschillende patronen waarnaar met zou willen minen. Deze verschillende patronen worden in deze sectie besproken.

Een eerste patroon waarin men geïnteresseerd zou kunnen zijn is de identificatie van verschillende klasses. De verschillende objecten worden dan gegroepeerd in verschillende nuttige subklasses. Klasse identificatie is een vorm van clustering.

Een tweede patroon is clustering op de gewone manier. Hiervoor kunnen de reeds bestaande algoritmen voor niet spatial clustering gebruikt worden, soms ook met enkele kleine aanpassingen om iets efficiënter te zijn. Het begrip afstandsfunctie moet hier ruim genomen worden, dus niet enkel de Euclidische afstand.

Een derde patroon kan noise bij in rekening brengen. Bij een verzameling van punten kan het zijn dat sommige punten zeer ver van de rest gelegen zijn. Deze punten kunnen ontstaan door meetfouten of gewoon enkele buitenbeentjes zijn. Deze punten noemt men noise. Het spreekt voor zich dat noise een negatieve invloed heeft op clustering (op sommige algoritmes meer dan andere) terwijl het niet echt wenselijk is dat deze punten in rekening gebracht worden. Sommige algoritmes zijn ongevoelig aan noise terwijl andere dan weer voorafgegaan kunnen worden door noise-verwijderende algoritmes.

Een vierde mogelijk patroon is het minen naar spatial association rules. Spatial association rules zijn van de vorm

$$P_1 \wedge \ldots \wedge P_m \rightarrow Q_1 \wedge \ldots \wedge Q_n(c\%).$$

c is de *confidence* van de regel, dit wil zeggen dat er $c\%$ kans is dat $Q_1 \wedge \ldots \wedge Q_n$ geldt wanneer $P_1 \wedge \ldots \wedge P_m$ geldt. Een voorbeeld van een association rule waar men geïnteresseerd is in de pure spatial eigenschappen (zonder bijkomende niet spatial eigenschappen) zou kunnen zijn:

$$is\_a(x, gas\_station) \rightarrow close\_to(x, highway)\ (80\%)$$

5.2 Algoritmes voor Spatial Mining

In deze sectie worden technieken en algoritmes voor het minen naar de in de vorige sectie vermelde patronen beschreven. Er zal ook gebruik gemaakt worden van de begrippen die werden geïntroduceerd in het vorige hoofdstuk.

5.2.1 Efficiënte klasse identificatie

Een eerste methode die besproken zal worden is de methode voor efficiënte klasse identificatie zoals voorgesteld door Ester, Kriegel en Xu [EKX95]. Het doel is hier om alle objecten van de database op te delen in verschillende subklassen. Het enige wat nodig is voor dit algoritme is een afstandsfunctie *dist*, er is geen domeinkennis nodig. Aan de hand van deze *dist* kan er dan geclusterd worden om de verschillende klasses te identificeren. Een clusteringalgoritme dat zeer goed werkt is het *k-means*-algoritme. *k-means* kiest in de initiële stap een gegeven k aantal punten, de *centroïdes* genaamd. Vervolgens herhaalt het de volgende twee stappen tot er niets meer wijzigt:

1. Voor elk object wordt het dichtstbijzijnde (volgens *dist*) centroïde berkend en wordt het aan die cluster toegevoegd.

2. Als alle objecten aan een cluster zijn toegevoegd wordt per cluster het gemiddelde (*mean*) berekend en worden deze gemiddeldes de nieuwe centroïdes.

Het resultaat is de clustering die niet meer wijzigt door het toepassen van vorige stappen.

Hoewel dit algoritme een goed werkend algortime blijkt te zijn, heeft het enkele nadelen op het gebied van spatial mining. Ten eerste is het algoritme

zeer gevoelig aan noise. Het gemiddelde wordt sterk beïnvloedt wanneer er één enkele waarde sterk verschillend is van de rest. Een tweede nadeel is dat het gemiddelde van een aantal objecten geen element is van de verzameling punten waarop geclusterd wordt, en dus weinig tot geen betekenis heeft. Ten slotte moet het gemiddelde van een verzameling punten ook gedefinieerd zijn, wat niet altijd het geval is, bijvoorbeeld wanneer de data niet numeriek is.

Een gelijkaardige aanpak aan *k-means* is *k-medoids*. Het algoritme is hetzelfde behalve dat er gebruik gemaakt wordt van representatieve objecten, de *medoids* genaamd, die dan wel deel uitmaken van de data set. De invloed van noise bij *k-medoids* is niet zo groot dan bij *k-means*. De medoids zijn wel elementen van de database en om medoids te bepalen is enkel de functie *dist* nodig.

Alvorens verder te gaan met het algoritme nog enkele notaties:

- O is de verzameling van objecten
- $M \subseteq O$ de verzameling van k medoids

Alle objecten zijn veelvlakken, voorgesteld als een verzameling van zijdes. De zijdes worden voorgesteld door de twee eindpunten. Zij $P \subseteq \mathbb{R}^3$ de verzameling van alle punten dan is het middelpunt van een object het wiskundig gemiddelde van de zijdes.

$$center : O \to P$$

De functie dist is gedefinieerd als volgt:

$$dist : P \times P \to \mathbb{R}_0^+ \tag{5.1}$$

maar kan uitgebreid worden voor objecten:

$dist : O \times O \to \mathbb{R}_0^+$

$dist(o_i, o_j) = dist(center(o_i), center(o_j))$

medoid is de functie die een object toewijst aan het dichtstbijzijnde medoid:

$medoid : O \to M$

$medoid(o) = m_i, m_i \in M, \forall m_j \in M : dist(o, m_i) \leq dist(o, m_j)$

Als maat voor de clustering (c) wordt de functie *total_distance* gebruikt (C_O is de verzameling van alle mogelijk clusteringen op O):

$total\_distance : C_O \to \mathbb{R}_0^+$

$total\_distance(c) = \sum_{m_i \in M} \sum_{o \in cluster(m_i)} dist(o, m_i)$

Het algoritme

PAM (Partitioning Around Medoids) is een algoritme van het type *k-medoids*. Het begint met een willekeurige verzameling van k medoids en herhaalt de stappen 1. en 2. zoals hierboven beschreven. Dit algoritme werkt zeer goed voor kleine datasets, maar voor grote databases wordt het algoritme zeer inefficiënt omdat het elk object met elk ander object gaat vergelijken.

Een beter algoritme, dat gebaseerd is op PAM, is *CLARANS* (Clustering Large Applications based on RANdomized Search). CLARANS maakt gebruik van een zoekheuristiek. PAM gaat alle vervangingen van medoid in niet medoids en andersom proberen om verbetering te krijgen, bij CLARANS wordt een verplaatsing onmiddellijk doorgevoerd wanneer het een verbetering blijkt te zijn. De clustering die bekomen wordt door één verplaatsing wordt een buur van de huidige clustering genoemd. Het aantal buren dat getest wordt, wordt bepaald door een parameter *maxneighbors* en de selectie van deze buren is willekeurig. Het proces dat de beste buur zoekt kan herhaald blijven worden tot er geen buur meer te vinden is die een verbetering van de *total_distance* is. Een clustering waarvan geen enkele buur meer beter is, wordt een lokaal optimum genoemd. Het aantal lokale optima dat gezocht moet worden, wordt bepaald door de parameter *numlocal*. Het CLARANS-algoritme is weergegeven in figuur 5.1.

Het hele algoritme begint met een for-lus die ervoor zorgt dat er *numlocal* lokale optima gezocht worden. Er wordt een klasse `clustering` verondersteld te bestaan met een aantal functies:

- *create_randomly(k)*: Creëert een willekeurige clustering met k clusters.
- *select_randomly(old,new)*: Selecteert willekeurig een medoid als *old* en een niet medoid als *new*.
- *calculate_distance_difference(old,new)*: Berekent het verschil in *total_distance* tussen de clustering met *old* als medoid en de clustering met *new* als medoid i.p.v *old*.
- *exchange(old,new)*: De keuze van *select_randomly* wordt effectief doorgevoerd.
- *calculate_total_distance()*: Berekent de *total_distance* voor de huidige clustering.

Het eerste wat gebeurt in één iteratie van het CLARANS algoritme is een willekeurige clustering kiezen. Vervolgens worden er willekeurige buren gekozen en dit maximaal *maxneighbor* keer. De teller j wordt gebruikt om bij te

```
clarans(int k,function dist,int numlocal,int maxneighbor)
 1: for (i = 1;i ≤ numlocal; i++) do
 2:    current.create_randomly(k);
 3:    j = 1;
 4:    while (j < maxneighbor) do
 5:       current.select_randomly(old,new);
 6:       diff = current.calculate_distance_difference(old,new);
 7:       if (diff > 0) then
 8:          current.exchange(old,new);
 9:          j = 1;
10:       else
11:          j++;
12:       end if
13:    end while
14:    dist = current.calculate_total_distance();
15:    if (dist < smallest_dist) then
16:       best = current;
17:       smallest_dist = dist;
18:    end if
19: end for
```

Figuur 5.1: Het CLARANS-algoritme

houden de hoeveelste buur geprobeerd wordt. Wanneer er een buur gevonden is die een verbetering is in de *total_distance*, dan wordt er doorgegaan met de buur en wordt teller j terug op 1 gezet omdat er weer opnieuw $maxneighbors$ geprobeerd mogen worden. Wanneer de gekozen buur niet verbeterd, zal de teller opgehoogd worden en zal er gewoon verder gegaan worden. Op een bepaald moment zullen er geen buren meer zijn die een verbetering in de *total_distance* opleveren en is er een lokaal optimum gevonden. Het beste van de reeds gevonden lokale optima wordt bijgehouden in de variabele *best* en de *total_distance* hiervan wordt opgeslagen in de variabele *smallest_dist*. Wanneer er dan een lokaal optimum gevonden is, zal de *total_distance* vergeleken worden. Indien de *total_distance* van het gevonden optimum kleiner is dan *smallest_dist*, is het gevonden optimum het beste dat reeds gevonden is en zullen *best* en *smallest_dist* de huidige clustering en de *total_distance* ervan worden.

De kost van het algoritme wordt grotendeels bepaald door de I/O-kost omdat deze veel groter is dan de CPU-kost. De functies van de cluster-klasse bepalen deze I/O-kost. De k medoids kunnen in het geheugen opgeslagen worden, maar de niet medoids moeten van schijf gelezen worden. Zij c het gemiddeld aantal objecten dat per pagina ingelezen kan worden, dan kunnen de kosten van de verschillende functies bepaald worden.

- *create_randomly(k)*: Moet k medoids kiezen wat maximaal k pagina's kost.
- *select_randomly(old,new)*: Moet slechts één niet medoid lezen wat kan met één pagina.
- *calculate_distance_difference(old,new)*: Moet alle objecten lezen wat n/c pagina's kost (waarbij n het aantal objecten in de database is).
- *exchange(old,new)*: Doet enkel een update op de medoids, hiervoor is geen disk-I/O nodig vermits de medoids in het gebeugen zitten.
- *calculate_total_distance()*: Moet evenzeer alle objecten lezen, n/c pagina's.

Wegens de heuristieke aard van het algoritme is het niet mogelijk het aantal iteraties van de binnenste lus exact te bepalen. De kost van de operaties in de binnenste lus zullen het meest opgeroepen worden en doorslaggevend zijn. De enige operatie in de binnenste lus met een grote kost is *calculate_distance_difference*. De kost zal dus bepaald worden door het aantal keer dat deze kostelijke functie wordt opgeroepen.

Ester, Kriegel en Xu beschreven nog twee technieken om de kost te verlagen. Een eerste techniek bestaat erin het aantal objecten dat gebruikt wordt om de clusters te bepalen te verminderen. Een tweede techniek maakt gebruik van het feit dat niet alle objecten invloed hebben op het resultaat en gebruikt enkel relevante objecten om de efficiëntie te verbeteren. Deze twee technieken zullen hier niet verder besproken worden.

5.2.2 Spatial characterization

Zoals in in hoofdstuk 4 vermeld, zullen niet enkel de attributen van een object een rol spelen bij spatial data mining, maar zal er ook een invloed zijn door de nabijgelegen objecten van een object. Dit wordt toegepast in spatial characterization zoals beschreven door Ester, Frommelt, Kriegel en Sander [EFKS98]. Spatial characterization krijgt een verzameling van doelobjecten aan de hand van spatial en niet spatial eigenschappen die typisch zijn voor de doelobjecten maar niet voor de hele database. De taak van het algoritme is om alle $(attribuut, waarde)$-tupels of object types te vinden die veel frequenter voorkomen in de doelobjecten en hun buren dan in de rest van de database. De attributen zijn niet spatial attributen, het spatial aspect zit in het gebruik van de buren. De object types zijn types van objecten zoals bvb. bergen, rivieren, ... in een geografische database. Een regel die het resultaat is van een karakterisatie is van de vorm

$$target \Rightarrow p_1(n_1, freq\_fac_1) \wedge \ldots \wedge p_k(n_k, freq\_fac_k)$$

De regel betekent het volgende: p_i komt $freq\_fac_i$ keer meer voor in de verzameling van alle elementen uit targets uitgebreid met n_i buren dan voor de gehele database. Voor de definities van buren e.d worden de begrippen uit hoofdstuk 4 gebruikt.

Definitie 5.1. Zij $G^{DB}_{neighbor}$ een neighborhood-graaf, zij $targets$ een deelverzamling van DB, zij $freq^s(prop)$ het aantal voorkomens van de eigenschap $prop$ in de verzameling s, met s een deelverzameling van DB, en zij $card(s)$ het kardinaalgetal van s dan is de *frequentie-factor* van $prop$ t.o.v $targets$ en DB gedefinieerd als volgt:

$$f^{DB}_{targets}(prop) = \frac{freq^{targets}(prop)}{card(targets)} / \frac{freq^{DB}(prop)}{card(DB)}$$

Zij $significance$ en $proportion$ reële getallen, zij $max\text{-}neighbors$ een natuurlijk getal en zij $neighbors^i_G(s)$ de verzameling objecten bereikbaar in $G^{DB}_{neighbor}$ vanuit een element uit s door maximum i bogen te gebruiken dan is het doel van *spatial characterization* om elke eigenschap $prop$ en elk natuurlijk getal $n \leq max - neighbors$ te ontdekken zodanig dat zowel

1. de verzameling $objecten = neighbors_G^n(targets)$ als
2. de verzameling $objecten = neighbors_G^n(\{t\})$ voor minstens $proportion$ veel t's met $t \in targets$

voldoen aan de voorwaarde:

$$f_{objecten}^{DB}(prop) \geq significance$$

of

$$f_{objecten}^{DB}(prop) \leq \frac{1}{significance}$$

□

In 1. worden alle objecten uit $targets$ gelijk beschouwd, terwijl in 2. elk object apart beschouwd wordt. De constante $proportion$ is de minimun confidence van de karakterisering.

Het algoritme

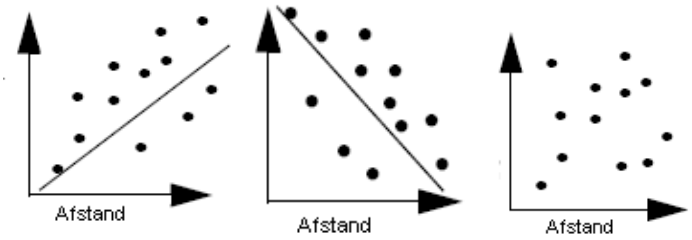
Het volledig algoritme is weergegeven in figuur 5.2. De input parameters zijn de neighborhood-graaf G_r^{DB}, de doelobjecten $targets$ en de waardes $significance$, $proportion$ en $max\_neighbors$ zoals hiervoor beschreven.

Het algoritme begint met het initialiseren van de $karakterisaties$ als de lege verzameling, een verzameling $regions$ als de verzameling $targets$ en een teller n op 0. Vervolgens worden de frequenties van alle $(attribuut, waarde)$-tupels berekend en opgeslagen. Hierna beginnen de iteraties die lopen voor $n = 0$ t.e.m $n = max\_neighbors$. Een iteratie van de while-lus begint met alle $(attribuut, waarde)$-tupels af te lopen en zal de frequentie-factoren t.o.v $targets$ berekenen (hiervoor kunnen de reeds berekende frequenties gebruikt worden). Indien een frequentie-factor aan één van de twee voorwaardes uit de definitie voldoet, wordt het tupel $(prop, n, f_{regions}^{DB}(prop))$ toegevoegd aan de $karakterisaties$. Wanneer alle mogelijke $prop$'s getest zijn, wordt $regions$ uitgebreid met de buren van elk object in de verzameling $regions$. Voor deze uitbreiding wordt de functie $neighbors$ gebruikt met als parameter de neighborhood-graaf, het wordt opgeroepen voor elk object in $regions$ en er wordt geen filter toegepast dus het derde element zal simpelweg $true$ zijn. Als laatste stap worden alle $(prop, n, f(prop))$-tupels geselecteerd wanneer $prop$ minstens $proportion$ keer voldaan is in de uitbreiding van de $targets$ met maximaal n stappen. Merk op dat n een belangrijke rol heeft omdat een $prop$ waar kan zijn in voldoende gevallen maar mogelijk niet meer waar is wanneer er nogmaals uitgebreid wordt. Uit de geselecteerde karakterisaties wordt dan uiteindelijk een regel gegenereerd en teruggegeven als resultaat.

characterization(graaf G_r^{DB}, verzameling van objecten *targets*, double *significance*, double *proportion*, int *max_neighbors*)

1: karakterisaties = $\emptyset$;
2: regions = targets;
3: n = 0;
4: bereken $frequency^{DB}$(prop) voor alle eigenschappen prop = (attribuut, waarde);
5: **while** n $\leq$ max_neighbors **do**
6: **for** elk attribuut = (attribuut uit DB of object type) **do**
7: **for** elke waarde van attribuut **do**
8: bereken $frequency^{regions}$(prop) voor elke prop = (attribuut,waarde)
9: **if** $f_{regions}^{DB}$(prop) $\geq$ *significance* OR $f_{regions}^{DB}$(prop) $\leq$ 1 / *significance* **then**
10: karakterisaties = karakterisaties $\cup$ (prop,n,$f_{regions}^{DB}$(prop));
11: **end if**
12: **end for**
13: **end for**
14: **if** n < max_neighbors **then**
15: **for** elk object in regions **do**
16: regions = regions $\cup$ neighbors(G_r^{DB}, object,TRUE)
17: **end for**
18: **end if**
19: n++;
20: **end while**
21: selecteer alle tupels (prop,n,f(prop)) uit karakterisaties die significant zijn voor minstens proportion veel regions met n uitbreidingen
22: **return** de regel gegenereerd uit deze karakterisaties

Figuur 5.2: Het karakterisatie-algoritme



(a) Positieve trend (b) Negatieve trend (c) Geen trend

Figuur 5.3: De verschillende trends

Voorbeeld 5.1. Een mogelijke regel die afgeleid zou kunnen worden uit een database met betrekking tot objecten waar een hoog percentage gepensioneerde mensen wonen:

$$\begin{aligned}&\text{hoogpercentage gepensioneerden} \Rightarrow \\ &\quad \text{appartementen per gebouw} = \text{zeer laag}(0, 9.1) \wedge \\ &\quad \text{percentage vreemdelingen} = \text{zeer laag}(0, 8.9) \wedge \\ &\quad \text{percentage gediplomeerden} = \text{matig}(0, 6.3) \wedge \\ &\quad \text{gemiddelde omvang van bedrijven} = \text{zeer laag}(0, 5.8).\end{aligned}$$

□

5.2.3 Spatial Trend Detection

Spatial trend detection maakt, in tegenstelling tot spatial characterization, gebruik van een ander aspect van neighborhood nl. neighborhood-paden. Spatial trend detection bestudeert de verandering van één of meerdere niet spatial attributen van een object dat zich weg beweegt van een gegeven start object. Het algoritme past regressie toe op de neighborhood-paden die zich van het startobject verwijderen. Het type regressie dat toegepast wordt is lineaire regressie omdat het efficiënt is en de invloed van de buren meestal lineair is of getransformeerd kan worden naar een lineair model. De afstand van het startobject wordt dan als onafhankelijke variabele gebruikt en de attribuutwaardes als afhankelijke variabele(n). Er kan een positieve trend, een negatieve trend of geen trend ontdekt worden, deze zijn grafisch weergegeven in figuur 5.3.

Definitie 5.2. Zij G een neighborhood-graaf, o een object-knoop, a een deelverzameling van alle niet spatial attributen, t een type functie (lineair

of exponentieel) gebruikt voor de regressie en $filter$ een van de filters voor het neighborhood-pad. Zij *min-conf* een reëel getal en *min-lenght* en *max-length* natuurlijke getallen, dan is de taak van spatial trend detection het ontdekken van een verzameling neighborhood-paden in G die vertrekken uit o en een trend van het type t in attributen a hebben met correlatie van minstens *min-conf*. Het pad moet aan $filter$ voldoen en de lengte moet tussen *min-length* en *max-length* liggen. □

De definitie laat toe om er twee verschillende interpretaties aan te geven. Het kan zijn dat alle ontdekte paden een trend van het gespecifieerde type bevatten of dat elk pad afzonderlijk een trend van het gespecifieerde type bevat. In het eerste geval spreekt men van *global trends* en in het tweede geval van *local trends*. Het is dan ook niet verwonderlijk dat er twee verschillende algoritmes zijn. Global-trend detection gaat detecteren dat de waardes van bepaalde attributen evenredig afnemen of toenemen naarmate er verder weg gegegaan wordt van een startobject. Local-trend detection detecteert dit ook, maar hier zal het enkel van toepassing zijn op één enkel pad.

Het global-trend algoritme

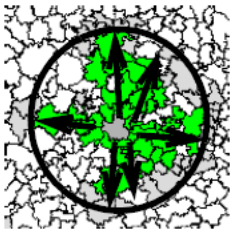
Het global-trend algoritme is weergegeven in 5.4. Het algoritme itereert over de lengte van de paden. Wanneer er op een bepaalde lengte nog een goede correlatie is zal het algoritme de paden uitbreiden, wanneer dit niet meer het geval is zal het de configuratie bij de laatste goede correlatie teruggeven en stoppen. Indien de maximum lengte van de paden bereikt wordt, zal de configuratie op dat moment worden teruggegeven.

De input voor het algoritme is een neighborhood-graaf g, een startobject o, de deelverzameling van alle attributen a, het type t dat lineair of exponentieel is, de mininmum confidence *min-conf*, de minimum en maximum lengte van van de paden *min-length* en *max-length* en de filter f die bvb. *variable-starlike* (zie definitie 4.8) kan zijn. De paden die gebruikt gaan worden, worden geïnitialiseerd op de paden van lengte maximum *min-length* vanuit het startobject o. Vervolgens worden er nog een aantal (logische) initialisaties gedaan. Het algoritme blijft lopen zolang de verzameling van paden (*paths*) waarmee gewerkt wordt niet leeg is. In elke iteratie van de while-lus worden alle paden afgelopen en van elk pad worden de objecten van positie $first\text{-}pos$ tot positie *last-pos* doorlopen. In de eerste iteratie zullen dit alle objecten zijn van alle paden, vanaf de volgende iteraties zijn dit telkens die bijgekomen objecten. Voor elk object o' wordt dan het verschil *diff* tussen het object $o$en o' in de attributen a berekend (er wordt aangenomen dat dit uitdrukbaar is) en de afstand *dist* tussen o en o'. Het koppel $(diff, dist)$ wordt dan toegevoegd

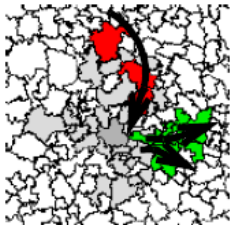
global-trend(graaf g, object o, verzameling attributen a, type t, double *min-conf*, int *min-length*, int *max-length*, filter f)

```
 1: paths = extensions(g,path(o),min-length,f);
 2: observations = ∅;
 3: last-correlation = 0;
 4: last-paths = ∅;
 5: first-pos = 1;
 6: last-pos = min-length;
 7: while paths != ∅ do
 8:    for elk path in paths do
 9:       for elk object van first-pos van path tot last-pos van path do
10:          diff = a(object) - a(o);
11:          dist = dist(object,o);
12:          observations = observations ∪ (diff,dist);
13:       end for
14:    end for
15:    voer regressie van het type t uit op de verzameling observations
16:    if abs(correlation) ≥ min-conf then
17:       last-correlation = correlation;
18:       last-paths = paths;
19:       if length(paths) < max-length then
20:          paths = extensions(g,paths,1,f);
21:          first-pos = last-pos;
22:          last-pos++;
23:       else
24:          paths = ∅;
25:       end if
26:    else
27:       return (last-correlation,last-paths);
28:    end if
29: end while
30: return (last-correlation,last-paths);
```

Figuur 5.4: Het global-trend detection algoritme



(a) Global-trend detection



(b) Local-trend detection

Figuur 5.5: Voorbeelden van global en local trends

aan de verzameling van observaties *observations*. Wanneer alle objecten behandeld zijn, wordt er regressie van het type t toegepast op de verzameling van observaties. Als de absolute waarde van de correlatie bekomen uit de regressie groter is dan de mininum confidence, dan wordt de correlatie-waarde opgeslagen als laatste goede correlatie-waarde *last-correlation* en de daarbijhorende paden als *last-paths*. Als de lengte van de paden nog strikt kleiner is dan *max-length* dan zullen de paden uitgebreid worden met segmenten van lengte 1 aan de hand van de extensions-functie, indien dit niet het geval is stopt de while-lus. Indien echter de correlatie-waarde kleiner zou zijn dan de minimum confidence dan wordt de laatst gevonden goede correlatie-waarde en de verzameling van daarbijhorende paden teruggegeven. Indien de while-lus afgelopen is en er nog niets is teruggegeven zullen *last-correlation* en *last-paths* teruggegeven worden als beste resultaat.

Voorbeeld 5.2. Stel dat er data beschikbaar is over huurprijzen van appartementen en huizen in een bepaalde stad inclusief buitenwijken en buurtgemeentes. Een mogelijke globale trend die ontdekt zou kunnen worden is dat hoe verder weg van het centrum hoe goedkoper de huurprijzen worden. Een grafische weergave is getoond in figuur 5.5(a) ([EFKS98]).

□

Het local-trend algoritme

Het local-trend algoritme is weergegeven in figuur 5.6. Het algoritme gaat in dit geval pad per pad werken. Het begint ook met de paden met lengte *min-length* en probeert aan de hand van regressie trends te ontdekken, wanneer dit lukt wordt het pad uitgebreid, wanneer dit echter niet meer lukt wordt het pad verwijderd en wordt er verder gegaan met het volgende pad.

Het algoritme krijgt als input dezelfde parameters als het global-trend detection algoritme. De paden *paths* worden op net dezelfde manier geïnitiali-

local-trend(graaf g, object o, verzameling attributen a, type t, double *min-conf*, int *min-length*, int *max-length*, filter f)

```
1: paths = extensions(g,path(o),min-length,f);
2: positive_trends = ∅;
3: negative_trends = ∅;
4: while paths != ∅ do
5:    observations = ∅;
6:    pad = eerste element van paths;
7:    paden = paden - pad;
8:    for elk object van min-length tot het laatste object van pad do
9:       diff = a(object) - a(o);
10:      dist = dist(object,o);
11:      observations = observations ∪ (diff,dist);
12:   end for
13:   voer regressie van het type t uit op de verzameling observations
14:   if abs(correlation) ≥ min-conf then
15:      if correlation > 0 then
16:         positive_trends = positive_trends ∪ (path,correlation);
17:      else
18:         negative_trends = negative_trends ∪ (path,correlation);
19:      end if
20:      if length(path) < max-length then
21:         paths = extensions(g,path,1,f) ∪ paths;//vanvoor toevoegen aan
            paths
22:      end if
23:   end if
24: end while
25: return positive_trends en negative_trends;
```

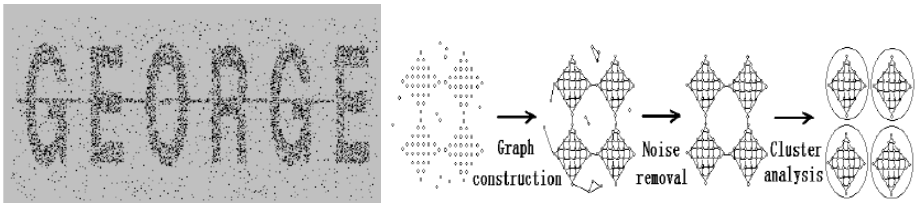
Figuur 5.6: Het local-trend detection algoritme

seerd, als de paden van lengte maximum *min-length*. Er worden twee (lege) verzamelingen aangemaakt: de positieve trends *positive_trends* en de negatieve trends *negative_trends*. Er wordt opnieuw door de paden gelopen tot er geen paden meer over zijn. Elke iteratie wordt er een verzameling observaties *observations* aangemaakt. Het eerste element wordt uit de verzameling paden gehaald en in de variabele pad gestoken. Vervolgens worden alle objecten van het pad vanaf positie *min-length* tot het einde van het pad doorlopen. Voor elk object worden de waardes *diff* en *dist* opnieuw berekend zoals in het global-trend algoritme en wordt het tupel $(diff, dist)$ toegevoegd aan de verzameling observaties. Wanneer alle object doorlopen zijn, wordt regressie van het type t toegepast. Als er een correlatie-waarde is die in absolute waarde groter is als de minimum confidence wordt het $(path, correlation)$-tupel opgeslagen. Indien het een positieve correlatie-waarde is, wordt het toegevoegd aan de verzameling *positive_trends*, indien niet, aan de verzameling *negative_trends*. Wanneer er een goede correlatie-waarde was, wordt het pad uitgebreid en terug vooraan toegevoegd aan *paths* zodanig dat het terug als pad geselecteerd zal worden in de volgende iteratie van de while-lus. Als er geen goede correlatie meer was, wordt het pad niet meer toegevoegd en zal de volgende iteratie verdergaan met het volgende pad. Merk op dat hierin een groot verschil zit met het global-trend detection algoritme. Wanneer er hier verder gegaan wordt met een volgend pad zal dit opnieuw een pad van minimale lengte zijn, terwijl in het global-trend algoritme alle paden steeds evenlang zijn. Het local-trend algoritme stopt wanneer elk pad ofwel de maximum lengte bereikt had en dus in laatste instantie niet meer aan paden toegevoegd werd, ofwel wanneer het geen trend meer bezit en dus ook niet meer toegevoegd werd.

Voorbeeld 5.3. Het local-trend detection algoritme kan uitgevoerd worden op dezelfde data als in het vorige voorbeeld. Het algoritme gaat dan een aantal paden vinden volgens dewelke de huurprijs achtereenvolgens daalt. Grafisch kan dit voorgesteld worden zoals in figuur 5.5(b). □

5.2.4 Efficiënte clustering met noise removal

In deze sectie wordt opnieuw een clustering methode besproken met een belangrijke bijkomende factor namelijk noise. Voor de meeste cluster-algoritmes is noise een zeer belangrijk aspect. Het idee bij clustering met noise verwijdering zoals voorgesteld door Qian en Zhang [QZ04a] is om, alvorens met het effectieve cluster-algoritme van start te gaan, eerst een algoritme toe te passen dat noise verwijdert. Het begrip noise is echter niet eenduidig gedefinieerd, daartoe wordt het begrip ambigue noise ingevoerd. Ambigue noise



(a) Een voorbeeld van ambigue noise

(b) De drie onderdelen van het CLEAN algoritme

Figuur 5.7: Ambigue noise en de stadia van het CLEAN algoritme

kan domein specifieke noise zijn, dit is noise die in het ene geval als noise kan bestempeld worden en in het andere geval niet. Een andere mogelijkheid is dat het om noise gaat die zeer dicht bij de echte data ligt, of dezelfde grootte en vorm heeft, en dus moeilijk of niet te detecteren is. Een voorbeeld van ambigue noise is weergegeven in figuur 5.7(a) ([QZ04a]). De horizontale lijn door *George* zou noise kunnen zijn, maar het zou net zo goed een stijl-element kunnen zijn. Het voorgesteld algoritme CLEAN (CLustering by Eliminating Ambiguous Noise) verwijdert eerst de noise alvorens clustering toe te passen. Redenen hiervoor zijn:

- De aanwezigheid van noise verstoort de clustering, het verhoogt de complexiteit en vergroot de kans op fouten.
- Vele cluster-algoritmes vereisen dat er een onderscheid gemaakt kan worden tussen noise en echte data.

Het CLEAN algoritme bestaat uit drie onderdelen:

1. Graaf constructie: De constructie van een k-wederkerige neighborhood-graaf.
2. Noise verwijdering: In dit stadium wordt de data opgedeeld in verschillende lagen door gebruik te maken van het k-core algoritme. Vervolgens worden de lagen aan de gebruiker getoond waarop deze ofwel de layer aanduidt die de grens vormt tussen data en noise (alles erboven is echte data, alles eronder is noise) ofwel alle layers aanduidt die al noise beschouwd moeten worden en de rest als echte data.
3. Clustering: Nadat de noise verwijderd is, wordt er een hiërarchisch cluster-algoritme toegepast om clusters van verschillende vormen en groottes te ontdekken.

De drie verschillende stadia van CLEAN zijn grafisch voorgesteld in figuur 5.7(b) ([QZ04a]).

Graaf-constructie

De eerste stap van het algoritme bestaat uit het construeren van de k-wederkerige neighborhood-graaf.

Definitie 5.3. Een *k-wederkerige neighborhood-graaf* is een graaf waarvan de knopen de verschillende objecten zijn. Er is een boog tussen knoop n_1 en knoop n_2 indien *zowel* n_1 één van de k dichtste buren van n_2 is als n_2 van n_1. □

De voordelen van het gebruik van deze graaf is ten eerste de isolatie van objecten die ver van de andere objecten gelegen zijn, ten tweede geeft de graaf de dichtheid van de objecten dynamisch weer en ten derde is het aantal bogen lineair in het aantal knopen, wat geen toename in de complexiteit veroorzaakt. De eerste twee voordelen zijn zeer nuttig bij het verwijderen van noise.

Het verwijderen van noise

Voor het verwijderen van noise wanneer de graaf is opgebouwd wordt het k-core algoritme toegepast. Alvorens met dit algoritme verder te gaan moet er een definitie voor een k-core gegeven worden.

Definitie 5.4. Zij $G = (V, E)$ een graaf met knopen V en bogen E dan is de subgraaf $H_k = (W, E|W)$ bekomen door de verzameling W een *k-core* of een *core van orde k* als en slechts als

$$\forall v \in W : degree(v) \geq k$$

en H_k de maximale subgraaf met deze eigenschap is (de functie $degree(v)$ telt het aantal bogen vertekkende uit v). □

De core van de grootste orde wordt de *main-core* genoemd. Cores hebben de volgende eigenschappen:

- Ze zijn genest: $\forall i < j \Rightarrow H_j \subseteq H_i$.
- Er bestaat een efficiënt algoritme om de core-hiërarchie te bepalen.
- Een core is niet noodzakelijk een volledig geconnecteerde subgraaf.

Het algoritme om de core-hiërarchie te bepalen gegeven een graaf G is rechttoe rechtaan. Verwijder alle knopen en de bogen eraan verbonden waarvan de *degree* kleiner is dan k. Om verwarring te vermijden wordt de k van k-wederkerige neighborhood-grafen k_m genoteerd en de k van k-cores als k_c.

Het k-core algoritme wordt dan gebruikt om noise te verwijderen. Zoals de definitie zegt moet elke knoop uit een core een *degree* groter hebben dan k_c, ook na het verwijderen van de objecten die hier niet aan voldoen. Dit wil zeggen dat het verwijderen van objecten die niet voldoen geen invloed heeft op de kwaliteit van de cores. Vermits de verbondenheid van de k-wederkerige graaf de dichtheid weergeeft, is het bepalen van k-cores net hetgene wat verwacht wordt als resultaat van het verwijderen van noise.

Tot nu toe zijn er reeds twee parameters die een invloed hebben op het resultaat: k_m en k_c. Onderzoek heeft aangetoond dat 90% van de noise verwijderd wordt met een k_m tussen 40 en 150 en 70% met een k_m tussen 20 en 200. Het bepalen van een goede k_c daarentegen vereist de nodige domeinkennis omdat het bepalen van het feit of data al dan niet noise is, domeinafhankelijk is. Om een k_c te bepalen zijn er twee mogelijkheden:

- de data representeren naar de gebruiker in verschillende lagen zodat de gebruiker een goede k_c kan aanduiden, of
- een k_c bepalen zoals dat gebeurt bij density-based clustering.

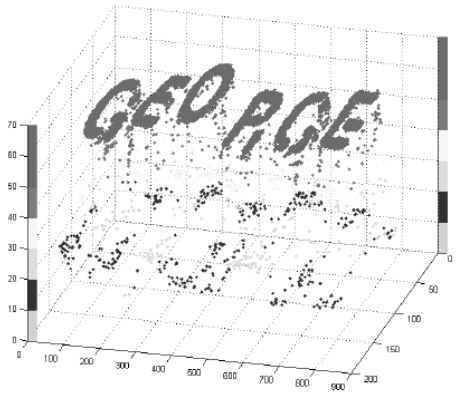
Beide manieren zullen kort besproken worden.

Bij de eerste manier worden de objecten opgedeeld in verschillende lagen. De waarde van k wordt als extra dimensie ingevoerd om de verschillende lagen te krijgen. Een object wordt enkel getekend in de hoogste laag waarvoor het nog in de k-core aanwezig is. De gebruiker kan vervolgens ofwel een laag aanduiden waarvoor alle lager gelegen lagen noise zijn en de hoger gelegen lagen echte data, ofwel kan de gebruiker alle lagen aanduiden die als noise beschouwd moeten worden.

Voorbeeld 5.4. Beschouw de verzameling objecten opgedeeld in de verschillende lagen zoals in figuur 5.8 ([QZ04a]). Aan de hand van deze visualisatie zou de gebruiker nu een laag kunnen kiezen waaronder alles eronder als noise beschouwd zou worden. Een mogelijke keuze voor deze laag zou kunnen zijn bij de laag $40 - 50$. Het zou echter ook mogelijk zijn dat de lijn zoals duidelijk zichtbaar in figuur 5.7(a) wel behouden moet blijven, dan zou het laag per laag selecteren het gewenste resultaat opleveren door de laag $10 - 20$ te behouden als echte data.

□

Een tweede mogelijkheid, buiten het visueel voorstellen van de verschillende lagen, is om gebruik te maken van de definitie van density-based noise. Het doel is om k_c te bepalen die de grens tussen data en noise definieert. Dit gebeurt als volgt:



Figuur 5.8: Visualisatie van de core-hiërarchie

Gegeven een dataset D, veronderstel dat de overeenkomstige k-wederkerige graaf $G = (V, E)$ is met v het aantal knopen en e het aantal bogen. Voer het k-core algoritme uit en noem de bekomen cores $H_0, H_1, \ldots, H_n$ waarbij H_i de core is van orde i. Bepaal $S_0, S_1, \ldots, S_n$ waarbij $S_n = H_n$, $S_{n-1} = H_{n-1} - H_n, \ldots, S_0 = H_0 - H_1$. De verzamelingen S_i bevatten op deze manier enkel de punten die op hetzelfde niveau van verbondenheid zitten maar niet de punten op een hoger niveau. Bijgevolg geldt ook $|S_0|+|S_1|+\cdots+|S_n| = v$.

Een aanname waaruit vertokken wordt is de aanname dat de data-punten van S_n echte data zijn, vermits dit de punten zijn die het best verbonden zijn. Deze aanname is redelijk want indien dit niet het geval was, zouden alle data-punten noise zijn. Het idee is vervolgens om deze verzameling uit te breiden met al de rest van de data die als echte data bestempeld kan worden. Er wordt vertrokken met als echte data S_n. Vervolgens wordt er telkens een core verder gekeken, indien deze core nauw verwant is met de echte data punten wordt de core toegevoegd. Indien dit niet het geval is, wordt gekeken of de core op zich voldoende groot en verbonden is. Als dit het geval is wordt de core alsnog toegevoegd en wordt er verder gegaan. Wanneer dit niet het geval is wordt er gestopt en zijn alle cores die de echte data voorstellen gevonden. Het volledige algoritme om de juiste k_c te vinden is weergegeven in figuur 5.9

S_i, C_i en D_i zijn respectievelijk *grootte* (size), *nabijheid* (closeness) en *dichtheid* (density). RC, RS en RD zijn respectievelijk *relatieve nabijheid*, *relatieve grootte* en *relatieve dichtheid*. Deze begrippen zijn gedefinieerd als volgt:

Definitie 5.5. De *nabijheid* tussen S_i en H_{i+1} is $C_i = \frac{E\_inter_i}{i|S_i|}$ en de *relatieve*

Find-k_c

```
1: for elke core S_i do
2:    Bereken |S_i|, |C_i| en |D_i|;
3: end for
4: Berken RS,RC en RD;
5: i = x;
6: while i ≥ 0 do
7:    if |C_i| > RC then
8:       continue;
9:    else
10:      if |S_i| > RS AND |D_i| > RD then
11:         continue;
12:      else
13:         return i;
14:      end if
15:   end if
16:   i–;
17: end while
```

Figuur 5.9: Het find-k_c-algoritme

nabijheid van alle n S-H-paren is gedefinieerd als:

$$RC = (\frac{1}{n}\sum_{i=0}^{n-1}|C_i|)P_c$$

waarbij E_inter_i het aantal bogen tussen S_i en H_{i+1} is en P_c een extra parameter. □

Definitie 5.6. De *relatieve grootte* van de $n+1$ verzamelingen S_i is gedefinieerd als:

$$RS = (\frac{1}{n+1}\sum_{i=0}^{n}|S_i|)P_s$$

waarbij P_s een extra parameter is. □

Definitie 5.7. De *dichtheid* van S_i is $D_i = \frac{|E_i|}{|S_i|}$ en de relatieve dichtheid van de $n+1$ S_i's is gedefinieerd als:

$$RD = (MAX(D_i) - MIN(D_i))P_d + MIN(D_i)$$

waarbij $0 \leq i \leq n$ en P_d een extra parameter is. □

De waardes P_c,P_s en P_d worden door een apart algoritme geleerd uit datasets van gelijke grootte en type.

Het algoritme gebruikt de relatieve waardes om te vergelijken of een core inderdaad nauw verwant of voldoende groot is. Het loopt alle cores af beginnend met S_n en kijkt of de volgende core voldoende nabijgelegen is door de closeness te vergelijken met de relative closeness. Indien dit geval is wordt de core toegevoegd aan de echte data en wordt er verdergegaan met de volgende core. Als dit niet het geval is, wordt getest of de core voldoende groot en dicht is door de grootte en de dichtheid van de core te vergelijken met de relatieve grootte en dichtheid van alle cores. Wanneer dit het geval is wordt de core alsnog toegevoegd aan de echte data en wordt er verdergegaan. Indien dit niet het geval was stopt het algoritme en is de scheiding tussen echte data en noise daar waar het algoritme stopt.

De clustering

Eens als er een onderscheid gemaakt is tussen noise en data, kan er een cluster-algoritme worden toegepast op de echte data. Als er geen noise meer is zullen vele cluster-algoritmes goede resultaten opleveren. Qian en Zhang [QZ04a] passen *GraphZip* ([QZ04b]) toe om het aantal punten te reduceren. GraphZip vervangt een reeks punten door één enkel punt om de efficiëntie van de clustering te verhogen. Eens de punten opgedeeld zijn kan er begonnen worden met het systematisch samenvoegen om zo tot een hiërarchische clustering te komen. Het samenvoegen gebeurt door een waarde $M(i, j)$ te berekenen voor elk paar van groepen van punten. Het paar met de grootste M wordt samengevoegd, dit net zolang totdat er één grote cluster overblijft. Voor het berekenen van M wordt er gebruik gemaakt van de initiële k-wederkerige neighborhood-graaf. Zij $G = (V, E)$ deze graaf en zij $S_1, S_2, \ldots, S_t$ de verzamelingen van punten die overeenkomen met de t initiële groepen. Zij $E(i, j)$ het aantal bogen tussen S_i en S_j en zij $|S_i|$ het aantal knopen in S_i dan is M gedefinieerd als volgt:

$$M(i, j) = \frac{E(i, j)^2}{\min\{|S_i|, |S_j|\}}$$

De formule verkiest het aantal bogen tussen twee groepen boven het aantal punten in een groep: hoe meer verbindingen hoe meer kans dat twee groepen worden samengevoegd.

Eens als de hiërarchische clustering klaar is, is het volledige CLEAN algoritme voltooid.

5.2.5 Mining naar spatial association rules

Het laatste algoritme dat voorgesteld wordt, is mining naar spatial association rules zoals voorgesteld door Koperski en Han [KH95]. Een spatial association rule of associatie-regel is een implicatie van een aantal spatial eigenschappen door een aantal andere spatial eigenschappen. Een associatie-regel is van de vorm

$$p_1(A_1) \wedge p_2(A_2) \wedge \ldots \wedge p_n(A_n) \Rightarrow q_1(B_1) \wedge q_2(B_2) \wedge \ldots \wedge q_m(B_m)$$

en heeft een bepaalde support en een bepaalde confidence. p_i en q_j zijn spatial of niet spatial predicaten en A_i en B_j zijn verzamelingen attributen met $1 \leq i \leq n$ en $1 \leq j \leq m$. De support drukt uit hoe dikwijls alle eigenschappen A_i en B_i samen voorkomen. De confidence drukt uit hoe groot de kans is dat alle B_i's voorkomen als alle A_i's voorkomen.

De gevolgen kunnen zowel spatial als niet spatial zijn. Er zijn dus meerdere gevallen mogelijk zoals bvb:

- Niet spatial gevolg uit spatial antecedenten.

$$is\_a(x, huis) \wedge close\_to(x, strand) \Rightarrow is\_duur(x)$$

- Spatial gevolg uit spatial/niet spatial antecedent(en).

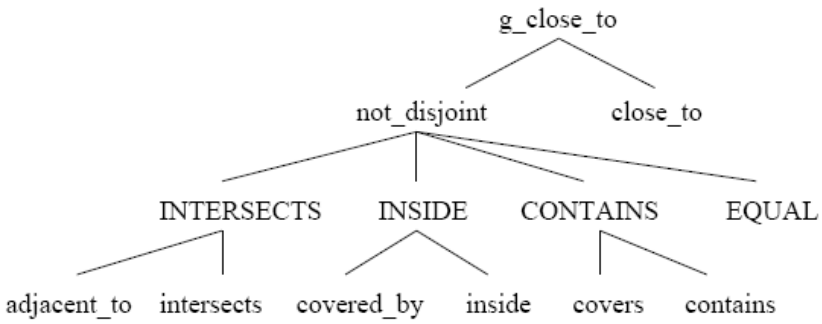
$$is\_a(x, pomp\_station) \Rightarrow close\_to(x, autosnelweg)$$

Als spatial predikaten zijn er verschillende mogelijkheden. Er kan gebruik gemaakt worden van de topologische relaties zoals besproken in 4.1.1 of van de richtingsrelaties zoals besproken in 4.1.3. Verder is het ook nog mogelijk andere, op afstand gebaseerde, relaties te gebruiken zoals bvb. *close_to* of *far_away*.

Eén enkel predikaat wordt een 1-predikaat genoemd. Een conjunctie van k predikaten wordt een k-predikaat genoemd.

Voorbeeld 5.5. Veronderstel dat het mining algoritme toegepast wordt op data in het relationeel model met volgende relaties:

1. $steden(naam, type, bevolking, geo, \ldots)$
2. $wegen(naam, type, geo, \ldots)$
3. $waterwegen(naam, type, geo, \ldots)$
4. $mijnen(naam, type, geo, \ldots)$



Figuur 5.10: De *g_close_to*-hiërarchie

5. $grenzen(naam, type, gebied1, gebied2, geo, \ldots)$

Alle relaties hebben een naam, een type en een geo. Het attribuut naam is triviaal. Het attribuut type wordt gebruikt om de tupels in categorieën te verdelen, bij wegen zou dit bijvoorbeeld $\{autosnelweg, snelweg, straat, laan, steeg\}$ kunnen zijn. Een query waarin men in geïnteresseerd zou kunnen zijn is: "zoek alle spatial association rules tussen grote steden en andere dichtbijgelegen objecten zoals mijnen, grenzen, waterwegen en belangrijke snelwegen". □

Het verdere verloop van het algoritme zal uitgelegd worden aan de hand van het voorbeeld.

De enige data die relevant is zijn steden van het type *groot*, wegen van het type *autosnelweg*, waterwegen van het type *zee*, *oceaan*, *groot_meer*, *grote_rivier* of *kanaal*, mijnen van elk type en grenzen van het type $landelijk$. Beschouw *g_close_to* als zijnde een algemene *close_to* functie die de predikaten zoals *adjacent_to* en *contains* omvat in een hiërarchische structuur zoals weergegeven in figuur 5.10.

Aan de hand van deze *g_close_to*-functie kan een tabel opgesteld worden waarin elke rij wil zeggen dat de stad uit de eerste kolom dichtbij (volgens *g_close_to*) de waardes in de andere kolommen ligt. Een voorbeeld van zo een tabel is gegeven in tabel 5.1.

Vervolgens wordt de support van elk tupel berekend en wanneer dit onder een door de gebruiker gespecifieerde minimum support ligt, zoals in tabel 5.1 het geval is met de kolom mijn, zal de kolom verwijderd worden. Op dit hoogste niveau van generalisatie kunnen reeds associatie-regels worden

Stad	Water	Weg	Grens	Mijn
Antwerpen	Schelde	E313	Nederland	
St.-Niklaas	Schelde	E313	Nederland	
Leuven		E314		
Hasselt	Albertkanaal	E314	Nederland	Winterslag
...	...	...	...	...

Tabel 5.1: De berekende *g_close_to*-relatie

Stad	Water	Weg	Grens
Antwerpen	<intersects,Schelde>	<intersects,E313>	<close_to,Nederland>
St.-Niklaas	<close_to,Schelde>	<close_to,E313>	<close_to,Nederland>
Leuven		<intersects,E314>	
Hasselt	<intersects,Albertkanaal>	<intersects,E314>	<close_to,Nederland>
...	...	...	...

Tabel 5.2: Specifieke spatial relaties tussen de objecten

afgeleid zoals bvb.:

$$\begin{aligned} is\_a(X, grote\_stad) &\Rightarrow g\_close\_to(X, water) \\ is\_a(X, grote\_stad) \wedge g\_close\_to(X, zee) &\Rightarrow g\_close\_to(X, nederlandse\_grens) \end{aligned} \tag{5.2}$$

De algemene *g_close_to*-relatie is niet de gewenste relatie om te gebruiken, daarom wordt de *g_close_to* relatie gespecifieerd tot op een concreter niveau in de hiërarchie, wat table 5.2 oplevert.

Tabel 5.2 vormt de basis voor de gedetailleerde berekening van de relaties op meerdere concept-niveau's.

De berekening begint op het hoogste niveau en berekent grote predikaten op dit niveau. Voor elke rij uit table 5.2 met een niet leeg water-attribuut wordt een teller voor water geincrementeerd. Al deze tellers samen leveren verschillende 1-predikaat rijen met bijbehorende support, weergegeven in table 5.3 als de rijen met $k = 1$. Wanneer de support kleiner is dan de gespecifieerde mininum support dan wordt de rij verwijderd. Stel dat de dataset uit 40 steden bestaat en de minimum support op dit niveau 50% bedraagt, dan worden in tabel 5.3 enkel rijen toegevoegd met een teller groter dan 20. De eerste rij wil zeggen dat er 32 van de 40 grote steden dichtbij water gelegen zijn.

De 2-predikaten worden berekend door de paarsgewijze combinatie van de 1-predikaten en de bijbehorende teller. Op dezelfde manier worden alle

k	k-predikaten	teller
1	<intersects,water>	32
1	<intersects,autosnelweg>	29
1	<close_to,autosnelweg>	29
1	<close_to,grens>	28
2	<intersects,water>,<intersects,autosnelweg>	25
2	<intersects,water>,<close_to,grens>	23
2	<close_to,grens>,<intersects,autosnelweg>	26
3	<intersects,water>,<close_to,grens>,<intersects,autosnelweg>	22

Tabel 5.3: k-predikaten verzameling op het hoogste niveau

k-predikaten berekend. De hogere k-predikaten staan ook weergegeven in tabel 5.3.

Vervolgens kunnen er reeds spatial association rules op een hoog niveau worden afgeleid uit tabel 5.3. Uit het tweede en het vijfde tupel kan de regel

$$is\_a(X, grote\_stad) \wedge intersects(X, autosnelweg) \Rightarrow intersects(X, water)$$

afgeleid worden met een support van 25 voor beide gedeeld door 29 voor enkel het antecedent ($25/29 = 86\%$). Merk op dat het predikaat $is\_a(X, grote\_stad)$ wordt toegevoegd omdat het altijd gaat over grote steden. Op dezelfde manier kan ook de regel

$$is\_a(X, grote\_stad) \wedge intersects(X, water) \Rightarrow close\_to(X, grens)(72\%)$$

afgeleid worden.

De andere k-predikaten verzamelingen op lagere niveau's worden op exact dezelfde manier berekend als hiervoor vermeld. De minimum support en mogelijk ook de confidence moeten echter wel verlaagd worden omdat de regels specifieker worden. Stel bvb. dat de minimum support van level 2 25% (teller ≥ 10) en van level 3 15% (teller ≥ 7). De resulterende tabellen worden weergegeven in tabellen 5.4 en 5.5.

Mogelijke regels die afgeleid kunnen worden uit de laatste twee tabllen zijn:

$$is\_a(X, grote\_stad) \Rightarrow intersects(X, grote\_rivier)(52, 2\%)$$

$$is\_a(X, grote\_stad) \wedge intersects(X, Schelde) \Rightarrow close\_to(X, Nederland)(78\%)$$

Merk op dat enkel de afstammelingen van de predikaten in de voorgaande tabellen verder beschouwd worden. k-predikaten die er op een hoger niveau niet doorkomen, zullen op een lager niveau dan ook niet beschouwd worden.

k	k-predikaten	teller
1	<intersects,grote_rivier>	21
1	<intersects,kanaal>	11
1	<close_to,Nederland>	28
1	<intersects,E-autosnelweg>	21
2	<intersects,grote_rivier>,<close_to,Nederland>	15
2	<close_to,Nederland>,<intersects,E-autosnelweg>	19
2	<intersects,grote_rivier>,<intersects,E-autosnelweg>	11
3	<intersects,grote_rivier>,<close_to,Nederland>,<intersects,E-autosnelweg>	10

Tabel 5.4: k-predikaten verzameling op het tweede niveau

k	k-predikaten	teller
1	<intersects,Schelde>	9
1	<intersects,Albertkanaal>	10
1	<close_to,Nederland>	28
2	<intersects,Schelde>,<close_to,Nederland>	7

Tabel 5.5: k-predikaten verzameling op het derde niveau

Een mogelijk alternatief is nog om ook het concept grote stad op te delen in verschillende niveau's om tot regels zoals bvb.

$$is\_a(X, hoofdstad) \wedge intersects(X, grote\_rivier) \Rightarrow close\_to(X, Nederland)$$

te komen.

De exacte details voor het implementeren van het algoritme zijn achterwege gelaten. Deze zijn terug te vinden in [KH95].

Hoofdstuk 6

Spatio-temporal Mining

In dit hoofdstuk zullen spatio-temporal mining technieken besproken worden. Eerst zullen de verschillende patronen waarnaar men zou willen minen kort toegelicht. Vervolgens worden deze verschillende mining-technieken apart besproken.

6.1 Spatio-Temporal Patronen

Er zijn twee belangrijke patronen waarin men mogelijk geïnteresseerd is. Een eerste soort patroon wat ook bij spatial mining aan bod komt is clustering. Wanneer men over spatio-temporal clusters spreekt zullen deze in weze hetzelfde zijn als gewone spatial clusters maar er zullen ook een aantal zaken anders geïnterpreteerd moeten worden. Het belangrijkste verschil is de afstand tussen twee moving points, het is niet meer mogelijk deze op een eenduidige manier te berekenen.

Bewegingspatronen is een andere belangrijke vorm van patronen waarin men geïnteresseerd kan zijn. Een mogelijk bewegingspatroon zou kunnen zijn dat een aantal punten zich op een relatief korte afstand van elkaar bevinden en allen in dezelfde richting bewegen. Een ander voorbeeld is een verzameling van bewegende punten die allemaal naar eenzelfde gebied aan het convergeren zijn.

De verschillende patronen zullen in de volgende secties uitvoerig besproken worden.

6.2 Spatio-temporal clustering

Een eerste data mining toepassing die ook toegepast wordt op het gebied van spatio-temporal data is clustering. Spatio-temporal clustering wordt

uitvoerig besproken door Mirco Nanni [Nan02]. Alvorens naar de toepassing en optimalistaties ervan te kijken worden de belangrijkste definities op een rijtje gezet.

Een heel belangrijk aspect bij elk type van clustering is het begrip *afstand*. Vanuit een theoretisch oogpunt is de beweging van een moving object o simpelweg een functie in de tijd. In de praktijk is dit echter niet zo. Het datamodel dat gebruikt wordt, is zoals in het trajectory location management nl. aan de hand van interpolatie. Van elk object wordt de locatie op een aantal tijdstippen doorgegeven en op de tijdstippen waarop de locatie niet bekend is, wordt er geëxtrapoleerd tussen de twee in tijd dichtsbijzijnde, wel gekende, locaties. De tijdstippen waarop de locaties van verschillende objecten geregistreerd worden, zijn ook niet allemaal synchroon. Op een tijdstip t kan bijvoorbeeld een locatie doorgegeven worden voor object o_1 en niet voor o_2, wat op een tijdstip t' net andersom zou kunnen zijn.

Een aanname die in vele cluster-algoritmes gedaan wordt, is dat er reeds een afstandsfunctie voorhanden is. Deze aanname is echter niet goed omdat het reeds een deel van het probleem uit de weg gaat, terwijl er ondersteuning moet zijn om een goede afstandsfunctie te vinden. Om met dit probleem om te gaan wordt er een algemeen afstandsschema gedefinieerd.

Definitie 6.1. Een *algemene spatio-temporele afstand* is een functie $d_a : O \times O \rightarrow \mathbb{R}^+$, met O de verzameling van alle objecten, die gedefinieerd is als volgt:

$$d_a(o_1, o_2) = \Phi_{t \in T}(d(o_1(t), o_2(t)))$$

waarbij $d(o_1(t), o_2(t))$ een afstandsmaat (parameter van het schema) voor de afstand tussen $o_1(t)$ en $o_2(t)$ is, $T \subset \mathbb{R}$ is het tijdsdomein van de moving points in de database en Φ is een functie die berekend wordt over de afstandsmaat d en domein T. □

Een voorbeeld van een afstandsmaat zou kunnen zijn de Euclidische afstand en een voorbeeld van een functie Φ zou het gemiddelde kunnen zijn. Het gemiddelde is een veel gebruikte functie voor Φ, andere mogelijkheden zijn o.a minimum, maximum, verschil tussen mininum en maximum.

Voorbeeld 6.1. Een clustering voor het vinden van moving points die gelijkaardige bewegingen maken maar niet noodzakelijk dicht bij elkaar gelegen zijn, kan gevonden worden door voor d de Euclidische afstand te kiezen en voor Φ het gemiddelde. Wanneer de onderlinge afstand op een bepaald tijdstip nooit sterk verschillend is van de afstand op andere tijdstippen dan maken ze dezelfde beweging en zullen ze tot dezelfde cluster behoren. □

Een belangrijk aspect dat niet over het hoofd gezien mag worden, is het feit dat een spatio-temporele afstandsmaat d_a moet voldoen aan de definitie van een metriek. Een metriek moet voldoen aan volgende eigenschappen:

1. $\forall i, j : d(i, j) = 0 \Leftrightarrow i = j$
2. $\forall i \neq j : d(i, j) > 0$
3. $\forall i, j : d(i, j) = d(j, i)$
4. $\forall i, j, k : d(i, j) + d(j, k) \geq d(i, k)$ (driehoeksongelijkheid)

Merk op dat 2. eigenlijk overbodig is, dit volgt uit 1. en het feit dat de doelruimte van d de verzameling $\mathbb{R}^+$ is. Uit de eigenschappen van een metriek kunnen dan, gegeven een standaard Φ, een aantal gevolgen getrokken worden.

Eigenschap 6.1. Zij $d(o_1(t), o_2(t))$ een metriek en $\Phi(f)|_T$ het gemiddelde, dan is ook d_a een metriek.

Eigenschap 6.2. Zij $d(o_1(t), o_2(t))$ een metriek en $\Phi(f)|_T$ het maximum, dan is ook d_a een metriek.

Eigenschap 6.3. Zij $d(o_1(t), o_2(t))$ een metriek en $\Phi(f)|_T$ het minimum of $max_T(f) - min_T(f)$, dan is d_a **geen** metriek.

In eigenschap 6.3 gelden eigenschappen 2 en 4 van een metriek niet. Wanneer twee objecten een verschillend traject afleggen, kunnen ze ergens gedurende het interval op dezelfde locatie zijn, wat op dat moment een afstand 0 oplevert. Het minimum zal dan 0 teruggeven wat in strijd is met de tweede eigenschap van een metriek. Voor de driehoeksongelijkheid kan van elk van de gevallen een tegenvoorbeeld gegeven worden.

- minimum: Beschouw 3 moving points o_1, o_2 en o_3 in $\mathbb{R}^1$. Beschouw $T = [0, 1]$, d de Euclidische afstand, $o_1(t) = 0$, $o_2(t) = t$ en $o_3(t) = 1$, dan geldt $d_a(o_1, o_2) = 0$, $d_a(o_2, o_3) = 0$ en $d_a(o_1, o_3) = 1$.
- (max - min): Beschouw 3 moving points o_1, o_2 en o_3 in R^2. Beschouw $T = [0, 1]$, d de Euclidische afstand, $o_1(t) = (0, 0)$, $o_2(t) = (cos(t), sin(t))$ en $o_3(t) = o_2(t) + \frac{1}{2}(cos(2t), sin(2t))$, dan geldt $d_a(o_1, o_2) = 0$, $d_a(o_2, o_3) = 0$ en $d_a(o_1, o_3) = 1$.

In beide gevallen zijn zowel $d_a(o_1, o_2)$ als $D(o_2, o_3)$ gelijk aan 0 en $D(o_1, o_3) = 1$ wat in strijd is met de driehoeksongelijkheid.

Als laatste is ook een lineaire combinatie van metrieken opnieuw een metriek.

Eigenschap 6.4. Zij $d_{a_1}, d_{a_2}, \ldots, d_{a_n} (n > 0)$ metrieken en $k_1, k_2, \ldots, k_n \in \mathbb{R}^+$ dan is ook $d_a(o_1, o_2) = k_1 d_{a_1}(o_1, o_2) + \ldots + k_n d_{a_n}(o_1, o_2)$ een metriek.

Een eerste methode die gebruikt wordt voor de clustering is de *dissimilarity matrix*. Een dissimilarity matrix is een matrix die alle onderlinge afstanden tussen alle mogelijke koppels van moving points bevat. Deze matrix wordt op voorhand berekend en wordt dan aan het cluster-algoritme doorgegeven. In sommige gevallen is er slechts enkel een matrix met de onderlinge afstanden. De algoritmes die deze matrix gebruiken, gebruiken ook meestal elke afstand wat wil zeggen dat elk element van de matrix berekend moet worden. Het spreekt voor zich dat het berekenen van al deze afstanden de grote kost is van het gehele cluster-algoritme.

Wanneer n groot wordt, zal het aantal afstanden dat berekend moet worden ook te groot worden en is er nood aan een andere techniek. Een betere techniek is *k-means* zoals reeds eerder beschreven. *k-means* beperkt zich tot het berekenen van een klein aantal van alle afstanden. Per iteratie wordt er voor elk punt de afstanden tot de k verschillende cluster-centra berekend, wat beduidend minder is dan bij een dissimilarity matrix.

6.3 Effeciënte mining naar bewegingspatronen

Indien er een dataset van een aantal moving points beschikbaar is die op een bepaalde locatie zijn op een aantal achtereenvolgende tijdstippen zou hierop gemined kunnen worden naar bewegingspatronen. Gudmundsson, van Kreveld en Speckman definiëren in [GvKS04] vier verschillende bewegingspatronen en leggen uit hoe hiernaar gemined kan worden. De bewegingspatronen en de algoritmes worden in de volgende secties besproken.

6.3.1 De gebruikte bewegingspatronen

De bewegingspatronen zijn gedefinieerd op een deelverzameling van alle moving points. Elk tijdstip wordt afzonderlijk bekeken waarbij voor elk punt de snelheid en de richting, die het op dat moment aanhoudt, bekend is.

Een eerste patroon dat gegeven wordt, is het patroon *flock*. Het flock-patroon houdt in dat een aantal moving points in dezelfde richting bewegen en dat ze dicht bij elkaar gelegen zijn. Dicht bij elkaar gelegen is gedefinieerd als zijnde binnen een cirkel met een welbepaalde straal r. De formele definitie luidt als volgt:

Definitie 6.2. Zij $m > 1$ een natuurlijk getal en $r > 0$ een reëel getal dan is een verzameling van minstens m moving points een *flock* indien de moving points zich binnen een cirkel met straal r bevinden en in dezelfde richting bewegen. □

Een tweede patroon is het *leadership*-patroon. Het leadership-patroon sluit nauw aan bij het flock-patroon. Leadership drukt hetzelfde uit als flock, behalve dat er een extra voorwaarde bijkomt die zegt dat één van de objecten (de leader) reeds een aantal stappen in de richting van de flock aan het bewegen was.

Definitie 6.3. Zij $m > 1$ en $\tau > 0$ twee natuurlijke getallen en $r > 0$ een reëel getal dan is een verzameling van minstens m moving points een *leadership*-patroon indien de moving points zich binnen een cirkel met straal r bevinden, in dezelfde richting bewegen en er één van de moving points reeds minstens τ tijdstippen in dezelfde richting aan het bewegen was. □

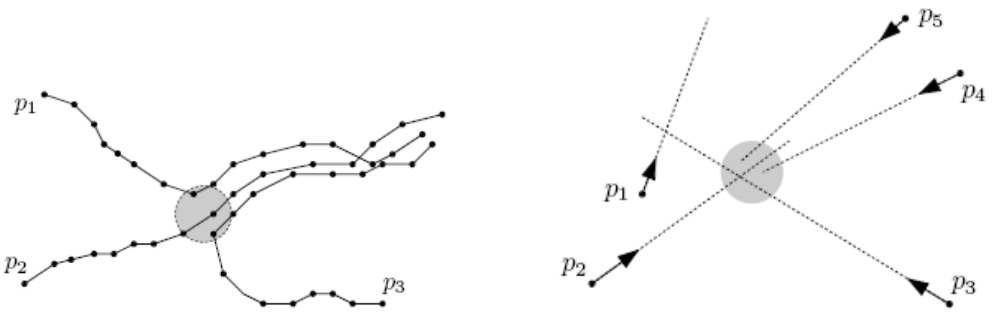
Een derde patroon is het convergentie-patroon. Het convergentie-patroon drukt uit dat een deelverzameling van de moving points zich naar dezelfde locatie begeven. Ook hier wordt 'dezelfde locatie' uitgedrukt als zich binnen een cirkel met een bepaalde straal bevinden. Het patroon wordt evenzeer op elk tijdstip gezocht, het is dus niet zo dat objecten per definitie ook effectief op die plaats zullen komen.

Definitie 6.4. Zij $m > 1$ een natuurlijk getal en $r > 0$ een reëel getal dan is een *convergentie*-patroon een verzameling van minstens m moving points die door een cirkel van straal r gaan passeren mits ze in dezelfde richting zouden blijven bewegen. □

Het vierde en laatste patroon is het *encounter*-patroon. Het encounter-patroon is een uitbreiding van het convergentie-patroon. Het encounter patroon drukt uit dat een aantal punten convergeren en op hetzelfde tijdstip op dezelfde locatie zullen zijn. Opnieuw geldt dit hier enkel wanneer de objecten hun snelheid en richting zouden behouden, wat natuurlijk niet altijd het geval is.

Definitie 6.5. Zij $m > 1$ een natuurlijk getal en $r > 0$ een reëel getal dan is een deelverzameling van minstens m moving points een *encounter*-patroon als de moving points zich op hetzelfde tijdstip binnen een straal van cirkel r zullen bevinden in de veronderstelling dat ze hun richting en snelheid houden. □

De eerste drie patronen worden grafisch weergegeven in figuren 6.1(a) en 6.1(b) ([GvKS04]).



(a) Flock-patroon op tijdstip 8 alsook een leadership-patroon met p_2 als leider

(b) Een convergentie-patroon van moving points p_2, p_3, p_4 en p_5

Figuur 6.1: Een flock en een convergentie-patroon

Het spreekt voor zich dat deze patronen enkel interessant zijn wanneer het om een groot aantal moving points gaat en een klein cirkelvormig gebied. Het zou echter ook niet nuttig zijn om hier concrete getallen op te plakken omdat bvb. het samenkomen van 50 auto's in een cirkel met straal $20m$ even interessant is als het samenkomen van 51 auto's in een cirkel met straal $19m$. Het is daarom beter om een benaderingsalgoritme te gebruiken waarbij m en/of r niet vastliggen. In plaats van een exacte r te gebruiken zal er een gebied gezocht worden met een straal $(1 + \varepsilon)r$. Het benaderen van m wil zeggen dat alle patronen van grootte $(1 + \varepsilon)m$ gevonden zullen worden, de patronen met een grootte tussen m en $(1 + \varepsilon)m$ gevonden *kunnen* (maar is niet zeker) worden en de patronen van grootte kleinder dan m niet gevonden zullen worden.

6.3.2 Het flock-algoritme

In deze sectie wordt een benaderingsalgoritme voor het flock-patroon besproken. Het leadership-algoritme zal in de volgende sectie besproken worden, maar het spreekt voor zich dat dit maar een kleine uitbreiding van het flock-algoritme zal zijn. Als input voor flock is er een verzameling van n moving points, een straal r en een minimum grootte $m \leq n$ om een flock-patroon te vormen. Om te beginnen worden alle punten opgedeeld in acht verschillende richtingen en wordt er een benaderingsalgoritme geconstrueerd dat de straal van het cirkelvormige gebied benadert. De 8 verschillende richtingen worden bekomen door de hoek van 2π op te delen in 8 gelijke stukken.

De basisbouwsteen voor het algoritme is een *quadtree*. Gegeven een verzameling moving points $S = \{p_1, p_2, \ldots, p_n\}$ gelegen in een vierkant C met

lengte l wordt een quadtree voor S recursief opgesteld als volgt: de root komt overeen met het vierkant C en heeft vier kinderen die overeenkomen met de opdeling van C in vier vierkanten van lengte $l/2$. Dit proces blijft zich herhalen tot op het punt dat elke tak tot een blad leidt. Een kind van een knoop is een blad wanneer het slechts één moving point bevat.

Veronderstel een verzameling S van moving points die allen in één van de acht richtingen bewegen. Het enigste wat dan nog rest is het bepalen van cirkels met straal r waarin minimum m moving points liggen. S kan dan simpelweg bekeken worden als een verzameling van punten in het vlak.

In eerste instantie wordt er een quadtree opgebouwd tot op het niveau dat elk niet leeg vierkant uit de tree een lengte heeft van maximum $\frac{\varepsilon}{4}r$. De recursie zal dus niet stoppen wanneer er slechts één punt in een vierkant zit, maar wanneer het vierkant klein genoeg blijkt te zijn. Vervolgens wordt er voor elke niet lege cel C informatie over het aantal punten in C bijgehouden, genoteerd als S_C.

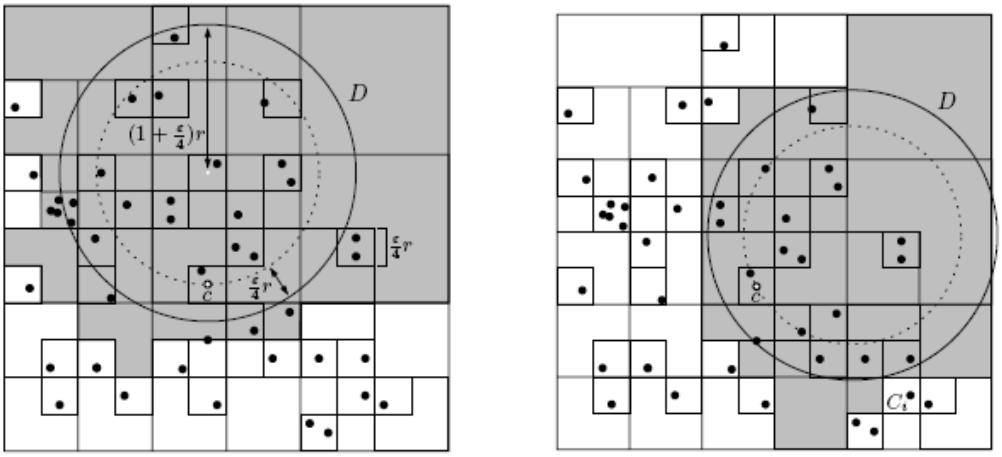
Gevolg 6.1. Een cirkel met straal $O(r)$ heeft een niet lege doorsnede met $O(1/\varepsilon^2)$ vierkantjes uit de quadtree.

Het voorgaande resultaat kan simpel berekend worden door de oppervlakte van de cirkel te delen door de oppervlakte van een vierkantje:

$$\frac{\pi r^2}{\frac{\varepsilon^2 r^2}{16}} = O(1/\varepsilon^2).$$

Het algoritme loopt alle niet lege cellen af. Het middelpunt van cel C wordt genoteerd als c. Alle punten binnen een afstand van $(2 + \frac{\varepsilon}{4})r$ tot c worden gezocht. Al deze punten liggen binnen *een* cirkel met straal $(1 + \frac{\varepsilon}{4})r$ waarvan c op een afstand r van het middelpunt ligt. Alle deze punten kunnen bekomen worden in tijd proportioneel tot het aantal cellen wat wegens het voorgaande gevolg $O(1/\varepsilon^2)$ bedraagt.

Wanneer deze punten bepaald zijn worden ze in een event queue gestopt. Vervolgens wordt er een cirkel D met straal $(1 + \frac{\varepsilon}{4})r$ geconstrueerd met c op een afstand r van het middelpunt en met het laagste punt van de cirkel op een afstand van $\frac{\varepsilon}{4}r$ tot c zoals weergegeven in figuur 6.2(a) ([GvKS04]). Merk op dat in de figuur ε voor de duidelijkheid groot gekozen is t.o.v de straal, in de praktijk zal de verhouding veel kleiner zijn. Om te beginnen wordt het aantal punten dat in één van de vierkanten met een niet lege doorsnede met D ligt berekend, genoteerd als S_D. Vervolgens wordt D rondgedraaid met c als middelpunt. Elke niet lege cel kan maximaal twee events genereren: binnenkomen in de cirkel D of buitengaan uit de cirkel D. Wanneer een cel C_i de cirkel binnenkomt wordt S_D verhoogd met S_{C_i}, indien een cel C_i de cirkel



(a) Initiële cirkel D (b) Cel C_i wordt toegevoegd aan D

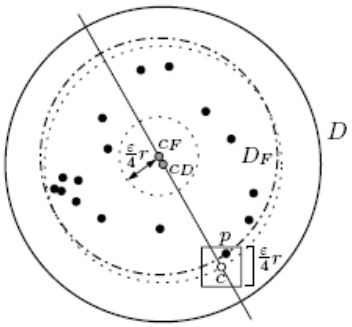
Figuur 6.2: Circulaire sweep van het Flock-algoritme

verlaat wordt S_D verlaagd met S_{C_i}. Wanneer $S_D \geq m$ geldt, wordt de cirkel D' met straal $(1+\varepsilon)r$ en met middelpunt het middelpunt van D teruggegeven als resultaat. Elke cel die een niet lege doorsnede met D heeft, ligt volledig binnen cirkel D' omdat de straal van D plus de lengte van de diagonaal van een vakje kleiner is dan de straal van D' $((1+\frac{\varepsilon}{4})r + \sqrt{2\frac{\varepsilon^2}{4}r^2} \leq (1+\varepsilon)r)$. De benadering van de straal schuilt hem dus in het feit dat de straal van de gevonden cirkels $(1+\varepsilon)r$ bedraagt.

Het enigte wat nog rest is laten zien dat alle flocks gevonden worden. Beschouw een verzameling F van moving points die een flock vormen. Er bestaat dus een cirkel D_F met straal r die F bevat en waarvan de omtrek een punt $p \in S$ bevat zoals weergegeven in figuur 6.3 ([GvKS04]). Zij C de cel die het punt p bevat en c het middelpunt van C. Vermits C niet leeg is zal c ooit als middelpunt gelden waarrond de cirkel D wordt gedraaid. Op een gegeven punt zal het middelpunt c_D van D op één lijn liggen met het middelpunt c_F van D_F. De afstand van c tot c_F is echter steeds kleinder dan of gelijk aan de lengte van een halve diagonaal van een cel plus de straal van D_F ($|cc_F| \leq \frac{\sqrt{2}\varepsilon}{8}r + r$) wat impliceert dat D_F volledig bevat zit in D. Vermits $S_D \geq m$ zal flock F dus ontdekt worden.

6.3.3 Het leadership-algoritme

Zoals reeds eerder vermeld is het leadership-algoritme slechts een uitbreiding van het flock-algoritme. Het flock-algoritme kan dan ook op een relatief eenvoudige manier worden uitgebreid om leadership-patronen te ontdekken.



Figuur 6.3: Flock F wordt ontdekt door het algoritme

Het algoritme begint, alvorens het flock-algoritme toe te passen, met het bepalen of een moving point reeds $\tau - 1$ tijdsstappen in dezelfde richting aan het gaan was. Voor elke cel in de quadtree wordt vervolgens ook opgeslagen of het al dan niet een leider bevat. Wanneer er vervolgens een flock ontdekt wordt, wordt er gekeken of er ook een leider in de flock zit. Indien dit het geval was, wordt het patroon gemeld als een leadership-patroon, indien dit niet het geval was gaat het algoritme gewoon verder.

6.3.4 Het convergence-algoritme

Voor het detecteren van een convergentie-patroon is de input opnieuw een verzameling van n moving points, een straal r en een minimum grootte $m \leq n$ voor de deelverzameling die het convergentie-patroon vormt. Wanneer er een lijn getrokken wordt vanuit elk punt in de richting waarin een moving point zich beweegt, wordt er een verzameling van half-rechtes bekomen zoals weergegeven in figuur 6.1(b) ([GvKS04]).

Het algoritme dat voorgesteld wordt door Gudmundsson, van Kreveld en Speckman in [GvKS04] is een benaderingsalgoritme op de grootte m van de deelverzameling die het patroon vormt. Het benaderingsalgoritme zoekt naar patronen met minstens $(1+\varepsilon)m$ moving points. Een gebied waar minder dan m punten voorbijkomen wordt niet gedetecteerd en een gebied met meer dan m en minder dan $(1+\varepsilon)m$ punten zou mogelijk gedetecteerd kunnen worden.

Om te beginnen wordt de voorstelling met de half-rechtes uitgebreid. Er wordt een evenwijdige met de rechte getekend aan beide zijdes en dit op een afstand r. Op die manier worden er halve strips bekomen van breedte $2r$.

Gegeven een verzameling van rechtes L en een parameter s, kan het vlak opgedeeld worden in t driehoeken $\Delta_1, \Delta_2, \ldots, \Delta_t$ zodat het inwendige van elke driehoek snijdt met maximum n/s rechtes uit L. Een verdeling zoals

deze wordt een $1/s$-cutting van L genoemd.

Beschouw nu opnieuw de verzameling halve strips behorend bij de moving points en noem deze H. Noem de verzameling van de $3n$ lijnen die de richtingen en de grenzen van de strips voorstellen L. Het algoritme begint met een initiële driehoek Δ die alle doorsnedes tussen de verschillende strips bevat. Het aantal strips die Δ volledig coveren, genoteerd als $|\Delta|$, is dan ook 0.

In een stap van het algoritme wordt er een driehoek Δ als input gegeven. Indien het aantal lijnen dat Δ snijdt groter is dan εm wordt er een $1/s$-cutting van Δ in t kleine driehoeken geconstrueerd (het kan aangetoond worden dat $t = O(s^2)$). Voor elke driehoek Δ wordt $|\Delta_i|$ berekend door $|\Delta|$ toe te voegen aan de strips die Δ snijden en Δ_i volledig bevatten. Indien het aantal lijnen dat Δ snijdt kleiner dan of gelijk aan εm is en $|\Delta| \geq m$ dan vormt elke cirkel met straal r en middelpunt in Δ een convergentie-patroon. Er wordt wel slechts één cirkel als resultaat gegeven per gevonden driehoek.

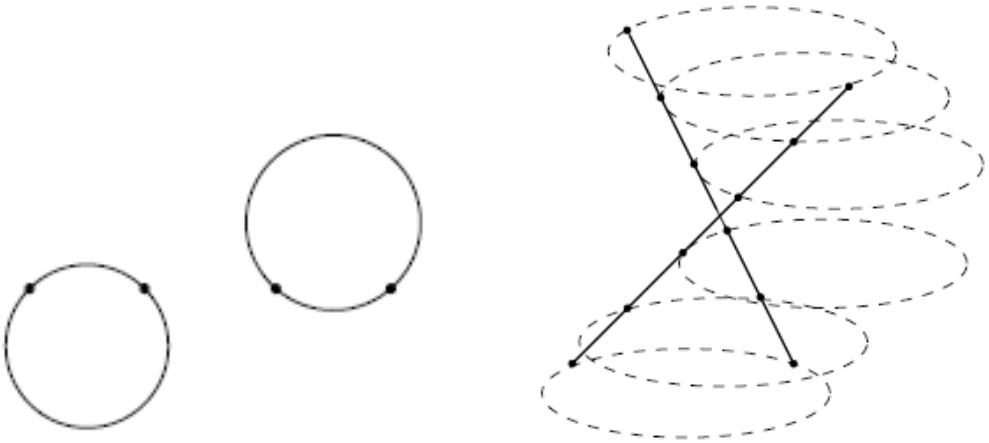
De reden waarom er een nieuwe cutting doorgevoerd wordt als er meer dan εm rechtes een driehoek snijden is omdat de punten waarvan er een intersectie is met de driehoek er geen zekerheid bestaat of het punt ook effectief tot het patroon zal behoren. De benadering zit hem dan ook in feit dat deze punten *mogelijk* tot het patroon behoren maar dat het niet zeker is. Net omwille van die reden mogen het er niet meer dan εm zijn want dit is het verschil tussen de exacte m en $(1 + \varepsilon)m$.

6.3.5 Het encounter-algoritme

In deze sectie wordt een algoritme om het encounter-patroon te ontdekken besproken. Het is een exact algoritme, met exact wordt bedoeld dat er geen benadering gedaan wordt van r noch m, dat alle patronen ontdekt. Net zoals voorheen is ook hier de input een verzameling van n moving points, een straal r en een grootte van de deelverzameling $m \leq n$.

In [GvKS04] worden nog drie andere algoritmes voor het encounter-patroon besproken. Het betreft hier twee andere exacte algoritmes ‘detecteer een patroon’ en ‘zoek het grootste’ en een benaderingsalgoritme op de straal.

Het probleem wordt gemodelleerd door tijd toe te voegen als een extra dimensie. Dit wil zeggen dat het probleem driedimensionaal wordt en dat de punten vertrekken bij $z = 0$ en omhoog bewegen doorheen de tijd. De projectie op het XY-vlak levert dan de trajecten van alle punten. Op elk tijdsttip wordt verondersteld dat de punten dezelfde richting en snelheid aanhouden, dus op elk tijdstip kan een moving point voorgesteld worden als een halve rechte.



(a) De twee mogelijke cirkels met straal r door twee punten

(b) Volume $V_{ij,1}$

Figuur 6.4: Twee mogelijke cirkels en het volumge $V_{ij,1}$ behorend bij het encounter-algoritme

Exact: zoek alle patronen

Voor het vinden van alle patronen worden de half-rechtes voorgesteld door cilinderachtige vormen door rond elk punt een cirkel te construeren met straal r. De bekomen figuur is geen echte cilinder maar zal wel indien het horizontaal gesneden wordt een cirkel van straal r opleveren.

Beschouw twee moving points op een tijdstip t dan kunnen zij op dat moment voorgesteld worden als twee half-rechtes met als beginpunten hun huidige locatie. Ze worden beide dan verondersteld dezelfde richting en snelheid aan te houden. Noem deze twee half-rechtes l_i en l_j. Er zijn op elk mogelijk tijdstip maximaal twee cirkels met straal r die een punt van l_i en l_j op hun omtrek hebben liggen (zie figuur 6.4(a) [GvKS04]). Deze twee cirkels bestaan enkel wanneer de twee punten op een afstand minder dan $2r$ van elkaar liggen. Vermits het om twee rechtes gaat is er slechts één interval waarin de afstand effectief kleiner kan zijn dan $2r$. Wanneer gedurende dit tijdsinterval achtereenvolgens de twee cirkels beschouwd worden, worden er twee cilindervormige volumes gecreëerd met een curve als as. Deze twee volumes worden $V_{ij,1}$ en V_{ij2} genoemd, een voorbeeld van deze volumes is gegeven in figuur 6.4(b).

Het algoritme gaat als volgt in z'n werk: neem $V = V_{ij,1}$ ($V_{ij,2}$ wordt nadien op exact dezelfde manier gedaan). Elke derde halve rechte l_h kan V maximaal twee keer snijden want het is een rechte en V is een gekromde cilinder. Wanneer, op een gegeven tijdstip V, gesneden wordt door $m - 2$

half-rechtes dan is er een encouter-patroon gevonden. De patronen worden effectief ontdekt door het bepalen van alle intersectie-intervallen met de $n-2$ andere half-rechtes en deze volgens tijd te rangschikken. Elk begin- of eindpunt van een interval zorgt ervoor dat er in het volgende tijdsinterval één moving point meer of minder binnen een afstand van r ligt. Wanneer er een deelverzameling van grootte m (inclusief de punten die V bepalen) gevonden wordt, is een encounter-patroon gevonden.

het volledige algoritme bestaat uit de toepassing van het voorgaande voor elke twee volumes van elk paar van half-rechtes.

6.4 Een clustering aanpak voor het ontdekken van persoonlijke geografische indexen

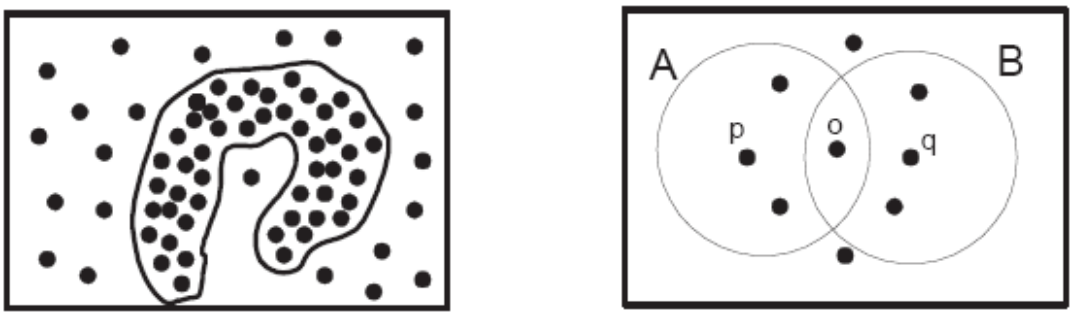
Een persoonlijke geografische index is een verzameling van belangrijke plaatsen voor een bepaald persoon, d.i., de plaatsen waar de persoon veel komt. Het minen naar geografische indexen is een probleem dat opgelost kan worden aan de hand van spatio-temporal clustering. Zhou, Frankowski, Ludford, Shekhar en Terveen stellen in [ZFL+04] een cluster aanpak voor, die gebruik maakt van density based clustering. De input voor het probleem is een verzameling van locaties en tijdstippen.

6.4.1 Eerdere pogingen

Er waren reeds een aantal andere pogingen om het probleem op te lossen. En eerste poging was een proefondervindelijk onderzoek. Het idee was om periodisch GPS-signalen binnen te krijgen van de locatie van een bepaalde persoon. Wanneer er geen signaal binnenkwam, betekende dit dat de persoon in een gebouw was. Achteraf worden de verschillende gebouwen waar de persoon geweest is aan hem of haar getoond en wordt er gevraagd ze te identificeren als belangrijke of onbelangrijke plaats. Er zijn aan deze aanpak echter twee grote nadelen: ten eerste zullen alleen gebouwen als interessante plaatsen worden ontdekt terwijl dit niet noodzakelijk het geval is en ten tweede is in vele gevallen een gebouw te algemeen.

Een andere aanpak was het gebruik van een *k-means*-algoritme zoals reeds eerder besproken. De som van de afstanden van alle punten tot één van de k cluster-centra wordt dan geminimaliseerd. Ook deze aanpak brengt een aantal nadelen met zich mee:

- Het aantal clusters moet op voorhand bekend zijn.



(a) Density based cluster (b) Twee density joinable clusters

Figuur 6.5: Density based clusters

- Alle punten worden behandeld wat als gevolg heeft dat het ook gevoelig wordt voor noise.
- Het algoritme is niet-deterministisch van aard. Dit houdt in dat de initiële keuze van de cluster-centra bepalend is voor het resultaat.
- De clusters zijn altijd cirkelvormig.

Een derde poging is een betere aanpak die ook de basis zal vormen voor het eigenlijk algoritme. De derde manier is het gebruik van density based clustering. Om density te definiëren worden er twee parameters gebruikt: Eps, de straal van een cirkel en $MinPts$ het minimum aantal punten in de cirkel. Density based clustering gaat ook verschillende clusters mergen die dicht bij elkaar liggen wat ervoor zorgt dat de clusters niet meer cirkel- of bolvormig zijn. Een voorbeeld van een density based cluster is weergegeven in figuur 6.5(a) ([ZFL+04]).

Density based clustering verhelpt veel van de nadelen van *k-means*. Ten eerste kan het zoals reeds vermeld clusters van een willekeurige vorm ontdekken, wat een goede verbetering is. Er is geen enkele reden om aan te nemen dat een cluster van interessante plaatsen van een persoon cirkelvormig is. Ten tweede wordt noise buiten beschouwing gelaten, het zal door de aard van het algoritme simpelweg niet opgenomen worden in een cluster. Bij het *k-means*-algoritme zou deze noise de clusters naar hun toe trekken wat hier niet het geval is. Ten derde is er een groot verschil in de parameters. Eps en $MinPts$ zijn niet afhankelijk van de input maar eerder afhankelijk van het feit van wat men wil minen. Een goede Eps en $MinPts$ kunnen bepaald worden die goed zijn voor elke uitvoering van het density based clustering algoritme. Het aantal clusters bij *k-means* is wel sterk afhankelijk van de data en moet door de gebruiker bepaald worden, wat zeker geen triviale opdracht

is. Het laatste nadeel dat verholpen wordt bij density based clustering is het niet-determinisme. Density based clustering geeft op een bepaalde input altijd hetzelfde resultaat.

6.4.2 DJ-clustering

Het eigenlijke algoritme heet *DJ-clustering* (*density-joinable clustering*) en is gebaseerd op density based clustering zoals beschreven in de vorige sectie. Het idee is als volgt:

- Bereken voor elk punt de buren (punten binnen een afstand Eps).
- Indien er minder dan $MinPts$ van deze buren zijn, wordt het punt gelabelt als noise.
- Indien het er minstens $MinPts$ en geen van de buren is reeds een cluster dan wordt een nieuwe cluster gecreëerd.
- Indien één van de buren wel een cluster was, dan worden alle punten binnen de afstand van Eps toegevoegd aan het reeds bestaande cluster.

De begrippen density based neighbors en density-joinable worden formeel gedefinieerd als volgt:

Definitie 6.6. De *density based neighbors* N van een moving point p, genoteerd als $N(p)$, zijn gedefinieerd als:

$$N(p) = \{q \in S | dist(p, q) \leq Eps\}$$

waarbij S de verzameling van alle moving points is. □

Definitie 6.7. $N(p)$ is *density-joinable* met $N(q)$, genoteerd als $J(N(p), N(q))$ indien er een punt o bestaat zodat zowel $N(p)$ als $N(q)$ het punt o bevatten. □

Een voorbeeld van twee density-joinable clusters is gegeven in figuur 6.5(b).

Gegeven deze twee definities kan een density en join based cluster ook formeel gedefinieerd worden.

Definitie 6.8. Een *density* en *join based cluster* C is gedefinieerd als volgt:

$$\forall p \in S, \forall q \in S, \exists N(p), N(q) : \exists J(N(p), N(q))$$

□

Pre-processing

Een eerste probleem dat opduikt bij deze manier van clustering is dat bvb. verkeerslichten waar een persoon dikwijls staat te wachten ook geregistreerd worden als interessante plaatsen. Om met dit probleem om te gaan kunnen vele verschillende filter-technieken toegepast worden. In dit geval worden er twee filters gebruikt. Een eerste filter houdt rekening met de snelheid. Snelheid is iets wat ook bepaald kan worden aan de hand van binnenkomende GPS-signalen. Wanneer een object niet stilstaat zal dit dan ook niet als een interessante plaats gerapporteerd worden. Een tweede filter gaat kijken of een locatie dicht bij de vorige locatie gelegen is. Indien dit het geval is wordt de locatie genegeerd. Dit heeft ook nog als voordeel dat het algoritme sneller zal gaan omdat er minder te processen punten zijn.

Het algoritme

Het uiteindelijk algoritme heeft volgende kenmerken:

- Elk punt is in een cluster of gelabelt als noise.
- Er is altijd minstens één punt in elke cluster.
- Het algoritme verdeeld de input in niet-hiërarchische clusters.
- De clusters overlappen niet.

en is weergegeven in figuur 6.6.

DJ-clusters(Eps,$MinPts$)

1: voeg de pre-processing door
2: kies een nog niet gebruikt punt p
3: **if** p is null **then**
4: **return**
5: **end if**
6: Bereken $N(p)$ met behulp van Eps en $MinPts$
7: **if** $N(p)$ is null **then**
8: label p als noise
9: **else**
10: **if** $N(p)$ is density joinable met minstens één andere cluster **then**
11: merge $N(p)$ met de andere clusters
12: **else**
13: creëer een cluster C gegeven $N(p)$
14: **end if**
15: **end if**
16: keer terug naar stap 2

Figuur 6.6: het DJ-cluster algoritme

Hoofdstuk 7

Toepassing van spatio-temporal data mining in de verkeerskunde

In dit hoofdstuk zullen de verschillende aspecten die reeds aan bod gekomen zijn in vorige hoofdstukken op een concreet domein worden toegepast namelijk verkeersproblemen. In sectie 7.1 zal het domein op zich worden toegelicht. Vervolgens zal in sectie 7.2 dieper ingegaan worden op hoe verkeer en verkeersstromen zich gedragen en wat de eigenschappen hiervan zijn. In sectie 7.3 zal besproken worden waar data mining aan bod kan komen binnen het domein en waar het ook een praktisch nut heeft. De secties 7.4 en 7.5 tenslotte, behandelen twee verschillende aspecten van het minen op verkeersstromen. In het eerste geval worden er conclusies getrokken nadat alles reeds heeft plaatsgevonden. In dit geval is er dus data voorhanden van bijvoorbeeld een volledige ochtendspits en worden hieruit dingen geleerd. Het tweede geval behandelt het real time geval. Er wordt hier niet gewacht tot alle data verzameld is, maar er wordt op het 'huidig' moment gemined. Het grote voordeel hiervan is uiteraard dat problemen vermeden kunnen worden.

7.1 Het domein verkeer en de problemen

Het verkeer is een interessant en concreet domein om spatio-temporal mining op toe te passen. Het probleem in het verkeer, dat het meest voor de hand ligt, is files. De meeste bestuurders kiezen gemakkelijkheidshalve de korste route van hun vertrekpunt tot aan hun bestemming. In vele gevallen is dit echter niet de meest efficiënte route. Een eerste nadeel van de kortste route is dat ze meestal niet via grotere wegen, wat sneller zou zijn, loopt. Dit is

wel een probleem wat de meeste bestuurders nog incalculeren. Een ander probleem is dat er dikwijls routes gekozen worden die passeren langs wegen waar het heel druk is en er dus traag verkeer is, simpelweg omdat dat de kortste route is. Als er sprake is van een echte file, die bijvoorbeeld via de radio of via GPS, is doorgegeven, dan passen bestuurders meestal hun route alsnog aan. Wanneer het echter over een drukte gaat, zullen zij hier meestal niet van op de hoogte zijn en zullen ze de drukte alleen maar verergeren.

Neem als concreet voorbeeld de ring rond Brussel. Dit voorbeeld zal ook als leidraad dienen doorheen dit en het volgende hoofdstuk. Stel dat een bestuurder met zijn wagen van Antwerpen naar Namen wilt rijden. De kortste weg zou zijn om de ring rond Brussel in wijzerzin te volgen omdat op die manier maar ongeveer een derde van de ring gereden moet worden. Het kan dan wel zijn dat het stuk ring dat de bestuurder neemt, veel drukker is dan de rest van de ring, het gedeelte dat hij dus niet neemt. Indien dit het geval is, is het mogelijk dat de ring in tegenwijzerzin volgen sneller zou zijn. Dit is een aspect waarvan bestuurders zich waarschijnlijk wel bewust zijn, maar het probleem is dat ze niet weten of het ook effectief beter zou zijn.

7.2 Gedrag van verkeer en verkeersstromen

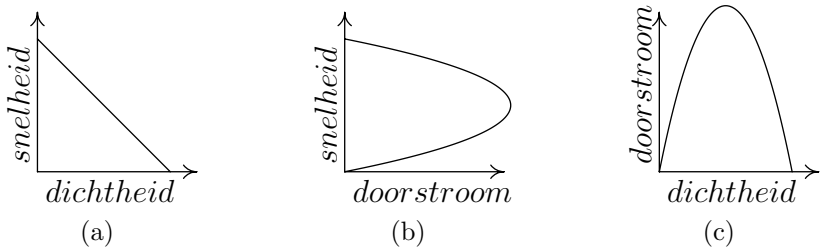
Om goed aan mining te kunnen doen is het belangrijk te weten hoe het verkeer zich gedraagt. In verband met verkeersstromen zijn er drie belangrijke parameters. Deze parameters worden bepsroken in [HCM00].

- *doorstroom*: Het aantal voertuigen dat passeert op een gegeven punt in een tijdsinterval korter dan een uur (typisch 15min).
- *snelheid*: Representatieve snelheid van een aantal voertuigen, meestal wordt het gemiddelde genomen van de individuele snelheden.
- *dichtheid*: Het aantal voertuigen op een bepaald gedeelte weg of autosnelweg.

De eenheden van *doorstroom*, *snelheid* en *dichtheid* zijn respectievelijk veh/h (voertuigen per uur), km/h en veh/km (voertuigen per kilometer). Er is een verband tussen alledrie de parameters nl.

$$dichtheid = \frac{doorstroom}{snelheid}$$

Aan deze formule kan gezien worden dat de drie afhankelijk van elkaar zijn. Twee van de drie parameters bepalen altijd de derde. De verklaringen van de parameters en de eenheden ervan deden dit ook reeds vermoeden.



Figuur 7.1: De relaties tussen de verschillende parameters

De verhoudingen tussen de verschillende parameters worden weergegeven in figuur 7.1.

In figuur 7.1(a) wordt het verband tussen de snelheid en de dichtheid weergegeven. Dit verband is vrij eenvoudig, wanneer de dichtheid groter wordt, neemt de snelheid af. De maximumsnelheid kan gereden worden als er geen voertuig op de baan is, wanneer de weg 'verzadigd' is bedraagt de snelheid, in theorie, 0 km/h.

De verhouding tussen snelheid een doorstroom is iets gecompliceerder, deze is weergegeven in figuur 7.1(b). De eenvoudigste manier om het verband te zien is door te vertrekken vanuit het punt waar de snelheid maximaal is. De snelheid is uiteraard maximaal wanneer de doorstroom minimaal is, want dan komt er geen enkel voertuig voorbij. Als vanuit dat punt de doorstroom toeneemt gaat de snelheid systematisch afnemen. Dit geldt voor de bovenste helft van de curve. Er komt echter een keerpunt op de top van de parabool. Vanaf een bepaald moment gaat de doorstroom zodanig toegenomen zijn dat het te druk wordt en de doorstroom terug gaat verminderen omdat de voertuigen niet meer voldoende snelheid kunnen halen. Dit keerpunt is de maximale doorstroom. De snelheid blijft nog steeds zakken en zal uiteindelijk 0 km/h worden. De doorstroom is dan ook helemaal teruggevallen tot 0 voertuigen per uur.

De laatste verhouding, weergegeven in figuur 7.1(c), is de verhouding tussen doorstroom en dichtheid. Deze verhouding kan voorgesteld worden als een bergparabool. De redenering die hierachter zit is eigenlijk dezelfde als bij het vorige geval. Als het aantal voertuigen per kilometer stijgt, dan stijgt ook de doorstroom. Vervolgens wordt hier hetzelfde keerpunt bereikt, waarna het aantal voertuigen per kilometer zal blijven stijgen, maar waar de doorstroom terug gaat afnemen omdat de drukte te groot is om nog vlot door te kunnen rijden.

Deze drie parameters zullen verder nog heel nuttig blijken voor het redeneren over verkeer en verkeersstromen. Omwille van de afhankelijkheid van

elke parameter van de twee andere, is hiet niet nodig alledrie de parameters in rekening te brengen maar volstaat het om er twee te gebruiken. De derde zal geen extra informatie toevoegen.

7.3 Waar data mining kan helpen

In de vorige secties werden problemen in het verkeer en bepaalde wetmatigheden in verband met verkeer besproken. De vraag die zich nu echter stelt is: "Waar kan data mining helpen?".

In sectie 7.1 werd reeds aangehaald dat er twee verschillende gevallen blijken te zijn waarbij data mining nuttig is. Een eerste geval waarbij reeds alle data gekend is, mining op verkeersdata van een verkeerssituatie die volledig achter de rug is. En een tweede geval dat real time data verwerkt en op het moment zelf bepaalde dingen leert, bijvoorbeeld het voorspellen van een file. Data mining kan in beide gevallen een nuttige bijdrage leveren.

Een eerste geval waarbij data mining na de feiten zeer nuttig kan zijn, is bij het opstellen van associatie-regels. Het is niet onwaarschijnlijk dat de ene file de andere impliceert. Stel in het geval van de ring rond Brussel dat er een file is in Leuven om half acht, dan is de kans groot dat er om half negen een file is op de ring rond Brussel in Tervuren. Een mogelijke regel die hieruit zou kunnen komen is de volgende:

$$\textit{traffic\_jam}(Leuven, 7h30) \Rightarrow \textit{traffic\_jam}(Tervuren, 8h30) \quad (70\%)$$

Uiteraard zullen files hierbij niet de enige zaken zijn die in rekening gebracht kunnen worden. Er zullen ook nuttige regels gevormd kunnen worden die betrekking hebben op bijvoorbeeld auto-ongevallen of wegwerkzaamheden. Het voordeel zal dan zijn wanneer een bestuurder zich in een situatie bevindt zoals het linkerdeel van een reeds eerder gevonden regel, dat hij/zij rekening kan houden met het gevolg van diezelfde regel. De regels moeten ook niet altijd negatief zijn, ze kunnen ook positieve zaken voorspellen.

Een tweede geval is mining naar drukke punten. Dit kan zowel real time als achteraf, maar hier is het real time geval uiteraard interessanter. Hiermee kan het probleem dat geschets wordt in de eerste sectie, het nemen van de ring langs de in afstand langere kant, opgelost worden. Bij deze aanpak zal gebruik gemaakt worden van dynamische routeplanning.

Deze twee aanpakken zullen in de volgende secties besproken worden.

7.4 Data mining na de feiten

Het belangrijkste feit bij mining op data die reeds volledig is, is dat er geen veronderstellingen of kansen gebruikt dienen te worden. Men moet geen rekening houden met vragen zoals 'Zou het kunnen dat...', 'Hoeveel kans is er dat...'. Dit brengt natuurlijk ook het nadeel met zich mee dat problemen ook pas bestudeerd kunnen worden nadat ze plaatsgevonden hebben. De reden om het toch te doen, is omdat de problemen dikwijls zullen terugkeren en dat ze in de toekomst vermeden kunnen worden. Zo zou men bijvoorbeeld kunnen afleiden uit data van twintig dagen met file in de ochtendspits dat het de 21e dag waarschijnlijk ook zo zal zijn.

Statistieken op basis van reeds gekende data

Een eerste, zeer eenvoudige, mogelijkheid om te minen[1] op de bekomen data is het opstellen van bepaalde statistieken. Enkele voorbeelden hiervan zijn:

- *Druktes*: De gemiddelde druktes op stukken van wegen, bijvoorbeeld tussen elke twee op elkaar volgende kilometerpalen. Er kan gekozen worden om deze druktes redelijk specifiek te berekenen ten opzichte van de tijd (bijvoorbeeld per kwartier), of om het zeer algemeen te houden (bijvoorbeeld per dag).

- *Snelheden*: Een andere mogelijkheid is de gemiddelde snelheden berekenen. Zoals reeds gezegd in de vorige sectie, houdt deze grootheid rechtstreeks verband met het vorige puntje. Ook hier kan gekozen worden om het zeer specifiek of minder specifiek te berekenen.

- *Ongevallen*: Een interessant gegeven wat niet zo zeer voor de hand ligt, is de kans dat er ongevallen gebeuren op bepaalde segmenten van een weg. Op deze manier is het mogelijk om gebieden te ontdekken waar regelmatig verkeersongevallen plaatsvinden, wat ook geen oninteressant gegeven is. Het kan ook nuttig zijn voor de overheid om te weten waar de zogenaamde 'zwarte punten' zich bevinden, want hier schort waarschijnlijk iets aan de infrastructuur of dienen er andere maatregelen getroffen te worden.

Dit zijn slechts enkele voorbeelden van zaken die uit de data geleerd kunnen worden. Hiervoor kunnen verschillende statistische technieken gebruikt worden. Vervolgens zou men aan routeplanning kunnen doen door de gewenste factoren in rekening te brengen als kost van de verschillende segmenten.

[1] De term minen is hier misschien niet echt op zijn plaats omdat het eigenlijk slechts een vorm van analyse van de data is.

In het geval van de ring rond Brussel zou dit als volgt in z'n werk kunnen gaan. Eerst wordt er gekozen in welke zin dat men voordeel wilt hebben, wil men sneller zijn, wil men minder kans op ongevallen hebben, ... Vervolgens zou al deze data op regelmatige tijdsintervallen, bijvoorbeeld wekelijks, geüpdate kunnen worden en zou op basis van de meest recente data aan routeplanning gedaan kunnen worden. Als men voor snelheid kiest, zou de vaststelling van het gebruikte routeringsalgoritme kunnen zijn dat het nemen van de ring via de langere weg voordeliger is.

Dit voorstel heeft echter wel een belangrijk nadeel. Neem even de situatie waarin iedereen deze methode zou toepassen in de ochtendspits, waar het meest waarschijnlijke criterium de snelheid zal zijn. Alle bestuurders, of hun routeplanners, merken dat ze weten uit het verleden dat er drukkere, en dus tragere, stukken zijn op de ring en minder drukke. Ze gaan echter allemaal bepalen of het vlotter is om de minder drukke stukken te nemen, maar daar wringt het schoentje. Deze minder drukke stukken zullen dan gekozen worden voor *alle* bestuurders wat ervoor zorgt dat de druktes zich verplaatsen naar deze minder drukke stukken en dat deze stukken dan minstens even druk worden. Het probleem is hier dan zelfs nog verergerd, want in afstand zal het traject langer worden en zullen ze er dus uiteindelijk nog langer over doen dan tevoren. Dit is duidelijk niet het gewenste resultaat! Het is echter wel zeer onwaarschijnlijk dat dit zal voorvallen.

Association rules

Een tweede mogelijkheid is om uit data informatie te halen in de vorm van association rules. De informatie die dan geleerd wordt is van de vorm: gegeven een aantal feiten, zijn een aantal andere feiten ook waar met een kans van $c\%$. Een aantal interessante vormen van regels om te minen worden in deze sectie besproken.

Een eerste vorm heeft betrekking op *files*. Wanneer er op een bepaalde plaats een file staat, is er veel kans dat er wat later een file staat op een verdere plaats. Deze regels zouden van volgende vorm zijn:

$$\mathit{traffic\_jam}(\mathit{locatie}_1, t_1) \wedge \mathit{extra\_cond}(\ldots) \Rightarrow \mathit{traffic\_jam}(\mathit{locatie}_2, t_2)\ (75\%)$$

met een bijbehorende confidence c en support s. De regel stelt dat wanneer er zich een file bevindt op locatie $\mathit{locatie}_1$ op tijdstip t_1 en een aantal extra condities voldaan zijn, zoals bijvoorbeeld het feit of het al dan niet een werkdag betreft, dat er dan $c\%$ kans bestaat dat er op tijdstip t_2 een file staat op locatie $\mathit{locatie}_2$.

Bestuurders kunnen hiermee rekening houden door te beslissen om $\mathit{locatie}_2$ te vermijden wanneer ze zich op $\mathit{locatie}_1$ bevinden op tijdstip t_1 en de ex-

tra condities voldaan zijn. Op deze manier kunnen ze ontsnappen aan de gebruikelijke files. Merk op dat het hier wenselijk is om te minen naar data van meerdere dagen om de toevalsfactor te elimineren. De redenen voor een file kunnen uiteenlopend zijn, maar wanneer de data van veel verschillende dagen afkomstig is, is de kans groot dat de regels betrekking hebben tot files die te wijten zijn aan de ochtendspits.

Een tweede interessante vorm heeft betrekking tot *verkeersongevallen*. Het spreekt voor zich dat verkeersongevallen een drastische invloed hebben op het verkeer. Er zijn veel verschillende vormen van regels denkbaar in verband met ongevallen. Een mogelijke regel zou kunnen zijn:

$$accident(Vilvoorde, 8h30) \wedge type\_of\_accident(volledig\_versperd) \Rightarrow traffic\_jam(Zaventem, 8h45) \quad (90\%) \quad (7.1)$$

Er zijn twee factoren die een belangrijke rol spelen bij verkeersongevallen. Een eerste is, dit wordt ook in regel 7.1 vermeld, het soort van ongeval. Bij een ongeval kan de weg gedeeltelijk of volledig versperd zijn, het kan een kettingbotsing zijn die een redelijke lengte heeft, ... Een tweede factor is het type van de weg. Als het een autosnelweg betreft zijn er meerdere rijstroken mogelijk en is er ook nog een pechstrook wat een groot verschil vormt met een ongeval op bijvoorbeeld een smalle secundaire weg.

Als er bepaalde regels geleerd zijn uit recente data, kunnen bestuurders hiermee geholpen worden door de file-propagatie door te geven. De informatie bij regel 7.1 die aan bestuurders gegeven kan worden, is dat ze de ring vanaf Vilvoorde richting Zaventem, en eventueel verder, best mijden.

Een volgende interessante vorm van associatie-regels, zijn regels in verband met *wegwerkzaamheden*. De redenering is volledig analoog als bij verkeersongevallen omdat de effecten hetzelfde zijn: filevorming, kan volledig of gedeeltelijke doorgang blokkeren, ... Het geval van wegwerkzaamheden is ook veel minder uitdagend in die zin dat er veel meer en op voorhand gekende informatie beschikbaar is. Er worden natuurlijk ook bij werkzaamheden omleidingen voorzien, maar wanneer de werken niet volledig de doorgang blokkeren, blijft het interessant om weten in welke mate ze files veroorzaken en tot op welke afstanden ervan.

Een laatste, die hier vermeld wordt althans, interessante vorm is in verband met de *zwarte punten*. Zwarte punten zijn bepaalde locaties waar veel ongevallen gebeuren. Het zou ook interessant zijn om bepaalde zaken te kunnen leren uit verkeersongevallen. Stel bijvoorbeeld dat ouders een veilige route willen hebben waarlangs ze hun kinderen naar school kunnen laten fietsen, dan willen zij uiteraard een route langs zo weinig mogelijk plaatsen waar frequent ongevallen gebeuren. Regels met betrekking tot verkeersongevallen

zullen vrij triviaal zijn. Zij zullen bijvoorbeeld uitdrukken dat indien het een schooldag is en het is tussen 8h 's morgens en 9h 's morgens dat de kans op een ongeval op een bepaalde locatie $c\%$ bedraagt.

De regels die gegeven een aantal feiten uitdrukken dat er veel kans is op een ongeval kunnen gebruikt worden om veilige routes te plannen. Dit kan gedaan worden door een routeringsalgoritme te gebruiken met zware gewichten indien er een redelijke kans is op een ongeval.

7.5 Real time data mining

Bij real time data mining is de situatie totaal anders. In plaats van gebruik te maken van data uit het verleden, is er nu enkel informatie over het huidige moment en het recente verleden. Het grote nadeel is uiteraard dat men niet in de toekomst kan kijken en dus vele aannames zal moeten doen. Het basisidee van wat in deze sectie besproken zal worden is dynamische routering. Als voorbeeld wordt opnieuw de ring rond Brussel genomen.

De meeste automobilisten kiezen vandaag de dag nog steeds voor de kortste route. Het grote probleem hierbij is uiteraard dat wanneer er veel volk naar Brussel rijdt, er altijd een aantal stukken zullen zijn waar het veel drukker is dan andere. De stukken die het drukst blijken te zijn, zijn de stukken tussen de afritten in Vilvoorde en Zaventem en tussen Zaventem en Tervuren. Dit is omdat voor vele bestuurders deze stukken van de ring op het kortste pad liggen.

Veronderstel voor de eenvoud dat alle voertuigen beschikken over een GPS-systeem dat periodisch de locatie van het voertuig doorgeeft. Neem verder ook aan dat de GPS-systemen aan routeplanning doen en dat de bestuurders dit ook volledig volgen, ze kunnen zich met andere woorden nooit vergissen. De route wordt voor het vertrek gepland. Indien alle bestuurders het korste pad kiezen, zullen er dus stukken zijn op de ring waar er file is en andere stukken waar er heel vlot verkeer is. In het vervolg van deze sectie worden betere routeringen voorgesteld.

Een eerste zaak die dient opgemerkt te worden is dat het kortste traject, zeker in de ochtendspits, niet het beste traject is. Het spreekt voor zich dat één kilometer afleggen in de file veel langer duurt dan tien kilometer afleggen tegen een gemiddelde snelheid van 110 km/h. Een eerste verbetering kan gevonden worden door snelheid in te calculeren.

Reistijd als kost

In sectie 7.2 werd reeds besproken hoe snelheid en dichtheid met elkaar verband houden. Dit feit kan in rekening gebracht worden bij het routeren naar een route die langer is in afstand, maar sneller zal zijn in tijd. Het idee is als volgt:

- De snelheid die gereden kan worden is afhankelijk van de dichtheid. Dit is een vaststaand feit dus dit patroon is terug te vinden in de snelheden die de auto's aanhouden. Aan de hand van deze dichtheid, die gekend is omdat alle voertuigen GPS-signalen doorsturen, kan bepaald worden hoe lang een wagen over een bepaald stuk autosnelweg zal doen.
- Beschouw deze tijd als de kost behorend bij een segment.
- Pas een routeringsalgoritme toe op de graaf van wegsegmenten en de tijdskosten in plaats van de afstand.

Het nadeel is echter dat de tijdskost een variabele kost is. Volgens de aanname die eerder gemaakt werd, wordt elke route voor het vertrek gepland. Wanneer een bestuurder bijvoorbeeld in Gent vertrekt, zal de kost van een stukje ring rond Brussel berekend worden op het moment dat de bestuurder nog in Gent is. Als de bestuurder na ongeveer een half uur op dit stukje ring arriveert, kan de verkeersituatie helemaal veranderd zijn.

Om met dit nadeel om te gaan, kan er getracht worden de beslissingen pas op een later tijdstip te nemen en zo ervoor te zorgen dat de gebruikte kost zo goed mogelijk de effectieve kost benadert.

Route herberekenen

Om een betere kost te gebruiken zal het dus nodig zijn om de beslissingen uit te stellen. De eenvoudigste manier om dit te doen is op elk kruispunt de situatie opnieuw te bekijken. Op elk kruispunt opnieuw kijken wat het kortste pad is naar de bestemming zal ervoor zorgen dat de weg die op dat moment ingeslagen moet worden het beste is. Vermits die weg dan ook op dat moment wordt genomen zal de kost die gebruikt is in de berekening ook een betere benadering zijn van de effectieve kost. Het zal dus altijd zo zijn dat enkel het eerste segment van het best gekozen pad ook met een zekerheid van 100% genomen wordt. De daaropvolgende segmenten kunnen zo mogelijk weer veranderen omdat de situatie op het volgende kruispunt terug veranderd kan zijn.

Op het eerste zicht lijkt dit proces veel kostelijker dan het vorige. Het enige verschil is echter dat het routeringsalgoritme meerdere malen moet worden uitgevoerd, maar dit is iets wat in de wagen zelf gebeurt. De updates

van de locaties van de verschillende voertuigen moet toch constant gebeuren vermits er op elk moment door elke bestuurder gerouteerd moet kunnen worden. Het telkens opnieuw berekenen van de routes is dus een te verwaarlozen extra kost.

Route berekenen op basis van druktevoorspelling

De routering gebaseerd op de tijd om een segment af te leggen, gecombineerd met herberekeningen op elk kruispunt, is een routeringsalgoritme dat al met vele factoren rekening houdt. Er zijn echter situaties waarin dat algoritme, en ook de voorgaande algoritmes, slechte besluiten nemen. Stel bijvoorbeeld de situatie waarin er enkel verkeer is vanuit Antwerpen en Gent. Vanuit Gent vertrekt er een groot aantal auto's en iets later vertrekken er ook een aantal auo's vanuit Antwerpen volgens de dynamische op tijd gebaseerde routering. Wanneer beide volgens het kortste pad de ring in tegenwijzerzin zouden nemen, gaat de drukte vanuit Gent net op de ring zitten wanneer de auto's uit Antwerpen aankomen op de ring. Het vorige algoritme kan dit ten vroegste doorhebben wanneereen auto aan het begin komt van het drukke segment van de ring, maar dan is het te laat want er is dan reeds gekozen om de ring in tegenwijzerzin te nemen en zal de kost om toch nog terug te draaien te groot worden. Merk op dat het enkel misgaat indien de drukte op de ring nog niet aanwezig is wanneer de route herberekend wordt aan het begin van de ring.

Om dit probleem mee in rekening te brengen moet er als het ware in de toekomst gekeken worden. Dit klinkt echter zeer onrealistisch, maar vermits de routes toch berekend worden aan de hand van data die voor alle auto's bijgehouden wordt, kunnen de verschillende geplande routes ook makkelijk onthouden worden. Dit zal het mogelijk maken om in de toekomst te kijken, omdat van elke auto geweten is hoe deze gaat rijden. Er wordt hier wel aangenomen dat de auto's deze route ook effectief gaan volgen. Hierdoor ontstaat de mogelijkheid om de situatie te bekijken hoe deze eruit zal zien wanneer een wagen in de toekomst op een bepaald kruispunt zal aankomen. Het grote voordeel is hier dan dat in de situatie die zonet beschreven werd, de auto's vanuit Antwerpen zullen merken dat er een drukte gaat komen op de ring op het tijdstip dat zij daar arriveren.

De zonet beschreven technieken worden geïmplementeerd, getest en besproken in het volgende hoofdstuk.

Hoofdstuk 8

Implementatie

In dit hoofdstuk wordt de gemaakte implemantie beschreven. De geschetste ideeën uit het vorige hoofdstuk in verband met dynamische routeplanning zijn geïmplementeerd en getest. De beschrijving van de implementatie en de resultaten van de testen worden in dit hoofdstuk besproken. In sectie 8.1 wordt het geconstrueerde model besproken. Vervolgens worden in sectie 8.2 de verschillende technieken die toegepast en getest werden besproken. Sectie 8.3 behandelt de bekomen resultaten. De laatste sectie, sectie 8.4, vermeldt extra informatie over de implementatie en de testen.

8.1 Het gebruikte model

Het model dat gebruikt werd is een modellering van de belangrijke autosnelwegen in België met centraal de ring rond Brussel. De bekomen graaf is weergegeven in figuur 8.1. De graaf is een gerichte graaf met telkens een boog in beide richtingen. Zo is de autosnelweg Lummen-Leuven totaal onafhankelijk van de autosnelweg Leuven-Lummen. Elke autosnelweg bestaat wel telkens in beide richtingen. Alle autosnelwegen worden benaderd door rechte lijnen tussen de knooppunten. De afstanden tussen de knooppunten zijn in verhouding met de realiteit. Elk stuk autosnelweg heeft een maximale snelheid. Op alle segmenten op de ring rond Brussel en alle segmenten van de halve ring rond Antwerpen geldt een snelheidsbeperking van 90 km/h. De overige autosnelwegen hebben een maximumsnelheid van 120 km/h.

De volgende stap is de auto's over het netwerk laten rijden. Iets wat belangrijk is bij het genereren van verkeersstromen, is het gebruiken van *bron-bestemmingsmatrices*. Een bron-bestemmingsmatrix definieert tussen elke twee knopen hoeveel auto's er van *bron* naar *bestemming* rijden. Welke bron-bestemmingsmatrix gebruikt wordt, hangt af van het testgeval. De

8.4 Implementatie- en testgegevens

De implementatie is volledig in Java (J2SE 1.5.0) gemaakt. De testen zijn uitgevoerd op twee machines, een Intel Pentium M 1.50 GHz 512 MB RAM en een Intel Pentium IV 1.50GHz 256 MB RAM. Het uitvoeren van de testen was dikwijls zeer intensief. De algoritmes *dist* en *dyn_time* toegepast op de toestroom naar Brussel duurden respectievelijk 40 minuten en 1 uur en 10 minuten. De testen op het enkel traject Antwerpen-Charleroi duurden allemaal tussen de 5 en de 10 minuten.

Hoofdstuk 9

Conclusie

In de vorige twee hoofdstukken zijn een aantal voorstellen voor data mining in de verkeerskunde en concrete uitwerkingen hiervan besproken. De gevonden technieken en de implementatie ervan werden getest op twee gevallen. Het eerste was van Antwerpen naar Charleroi rijden waar de ring in wijzerzin dan wel in tegenwijzerzin genomen kon worden. In tegenwijzerzin was de afstand iets korter, maar het is dikwijls efficiënter via de andere kant. Een goed algoritme moet dit ontdekken en indien nodig de alternatieve route kiezen. Het tweede geval was een vereenvoudigde voorstelling van het verkeer dat naar Brussel rijdt. Hier kwam vooral het probleem naar boven dat in Leuven veel verkeer samenkwam dat allemaal naar Brussel reed en dus file veroorzaakte.

Er werden vier verschillende algoritmes voorgesteld: *dist*, *time*, *dyn_time* en *prev*. Het algoritme *dist* liet alle auto's rijden volgens het kortste pad van vetrekpunt tot bestemming. Het algoritme *time* paste een routering toe op basis van de tijd die nodig was om een bepaald stuk weg af te leggen, gebaseerd op de drukte op dat stuk. Hierbij werd een route enkel voor het vertrek berekend. Omdat de tijd die nodig is om een weg af te leggen, gebaseerd op de drukte, constant verandert, werd het algoritme *dyn_time* gemaakt. Dit algoritme deed in weze hetzelfde, maar herberekende de route op elk kruispunt opnieuw. Dit idee werd nog verder doorgevoerd door zelfs in de toekomst te gaan kijken. Hierbij werd verondersteld dat de auto's hun route bleven aanhouden die zij in het begin gepland hadde. Dit algoritme werd *prev* genoemd.

Het algoritme *dist* bleek de slechtste resultaten te halen op het geval Antwerpen-Charleroi. Dit was uiteraard te verwachten. Toepassing van het *time*-algoritme op dezelfde situatie leverde een verbetering van de gemiddelde reistijd op van ongeveer 20 minuten. De volledige simulatie was uiteindelijk ook ongeveer een uur vlugger afgelopen. De verbetering die er nog eens extra

bijkwam door *dyn_time* toe te passen in plaats van *time*, was mathematisch niet zo groot. Het leverde een snelheidswinst op van een 4-tal minuten in gemiddelde reistijd. De volledige simulatie was een 7-tal minuten eerder uitgeklaard. Een belangrijke verbetering was echter wel de verdeling van de verkeersstromen. Bij *time* was er een alternatie tussen opstropping langs de ene kant van de ring en opstropping langs de andere kant van de ring. Bij *dyn_time* werd dit mooi verdeeld over de beide kanten van de ring. Het *prev* algoritme kan in dit geval geen verbetering meer brengen. Het algoritme *prev* zorgt voor een grote verbetering wanneer er file op komst is die op het moment dat er gepland wordt nog niet aanwezig is. Bij *dyn_time* kan dit enigszins opgevangen worden, maar dit kan nog verkeerd lopen wanneer een file te laat ontdekt wordt en terugdraaien niet meer efficiënt is. Bij *prev* wordt dit allemaal ingecalculeerd en dus ook tijdig ontdekt.

Bij het geval waarin de toestroom naar Brussel gemodelleerd werd, kan er niet meer zo eenvoudig een getal geplakt worden op de verbetering. Testen hebben uitgewezen dat er wel degelijk vele voordelen zijn van de routeringen op basis van tijd ten opzichte van de routeringen op basis van afstand. De simulatie waarin via kortste afstand gereden werd, duurde 3h 24m en de situatie waarin op tijd gebaseerd werd, duurde in totaal slechts 3h 2m. Een verbetering van meer dan 20 minuten dus. Het algoritme *prev* bleek in dit geval zeer snel door te hebben dat er file ging ontstaan in Leuven tot in Brussel. Dit werd reeds na 3 minuten ontdekt! Op dat moment was er zelfs nog geen sprake van een drukte.

De bekomen resultaten zijn bevredigend maar de testen waren ook redelijk artificieel. Ze deden wel duidelijk blijken dat de ideëen werkten, wat ook de bedoeling was. Het doel was niet om de hele fileproblematiek in België op te lossen!

Bibliografie

[EFKS98] Martin Ester, Alexander Frommelt, Hans-Peter Kriegel, and Jorg Sander. Algorithms for characterization and trend detection in spatial databases. In *Proceedings of 4th International Conference on Knowledge Discovery and Data Mining*, pages 44–50. AAAI Press, 1998.

[EGKS99] Martin Ester, Stefan Grundlach, Hans-Peter Kriegel, and Jorg Sander. Database primitives for spatial data mining. In *Datenbanksysteme in Buro, Technik und Wissenschaft*, pages 137–150, 1999.

[EKX95] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *SSD '95: Proceedings of the 4th International Symposium on Advances in Spatial Databases*, volume 951 of *Lecture Notes in Computer Science*, pages 67–82. Springer-Verlag, 1995.

[GvKS04] Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckman. Efficient detection of motion patterns in spatio-temporal data sets. In *ACM GIS 2004*, pages 250–257, 2004.

[HCM00] Highway capacity manual, 2000.

[KG05] Bart Kuijpers and Floris Geerts. Real algebraic geometry and constraint databases. manuscript., 2005.

[KH95] Krzysztof Koperski and Jiawei Han. Discovery of spatial association rules in geographic information databases. In *SSD '95: Proceedings of the 4th International Symposium on Advances in Spatial Databases*, volume 951 of *Lecture Notes in Computer Science*, pages 47–66, London, UK, 1995. Springer-Verlag.

[MSI02] Hoda Mokhtar, Jianwen Su, and Oscar Ibarra. On moving object queries: (extended abstract). In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 188–198. ACM Press, 2002.

[Nan02] Mirco Nanni. *Clustering methods for spatio-temporal data*. PhD thesis, Universita degli studi di Pisa, December 2002.

[QZ04a] Yu Qian and Kang Zhang. Discovering spatial patterns accurately with effective noise removal. In *DMKD '04: Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 43–50. ACM Press, 2004.

[QZ04b] Yu Qian and Kang Zhang. Graphzip: a fast and automatic compression method for spatial data clustering. In *SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 571–575. ACM, 2004.

[SXI01] Jianwen Su, Haiyan Xu, and Oscar H. Ibarra. Moving objects: Logical relationships and queries. In *SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, volume 2121 of *Lecture Notes in Computer Science*, pages 3–19. Springer-Verlag, 2001.

[TWZC02] Goce Trajcevski, Ouri Wolfson, Fengli Zhang, and Sam Chamberlain. The geometry of uncertainty in moving objects databases. In *EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology*, volume 2287 of *Lecture Notes in Computer Science*, pages 233–250. Springer-Verlag, 2002.

[Wol02] Ouri Wolfson. Moving objects information management: The database challenge. In *NGITS '02: Proceedings of the 5th International Workshop on Next Generation Information Technologies and Systems*, pages 75–89. Springer-Verlag, 2002.

[ZFL+04] Changqing Zhou, Dan Frankowski, Pamela Ludford, Shashi Shekhar, and Loren Terveen. Discovering personal gazetteers: an interactive clustering approach. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 266–273. ACM Press, 2004.

[ZSI02] Hongjun Zhu, Jianwen Su, and Oscar H. Ibarra. Trajectory queries and octagons in moving object databases. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 413–421. ACM Press, 2002.