Generaliseren van vlakkenpartities (1)

GAP-trees, theorie en implementatie[1])

geo-information engineering, spatial data structures, theory
geo-informatievoorziening, ruimtelijke gegevensstructuren, theorie
KEYWORDS
TREFWOORDEN

Dit artikel toont de resultaten van *on-the-fly* generaliseren met behulp van de GAP-tree ('Generalized Area Partitioning'-tree oftewel de 'Gegeneraliseerde Vlakken Partitie'-boom) toegepast op twee grootschalige topografische datasets, de vlakGBKN 1:1.000 en de Top10vector. Door toepassing van de GAP-tree is het nu mogelijk om interactief te navigeren door deze ruimtelijke datasets vanaf de oorspronkelijke schaal 1:1.000 (1:10.000) via een middenschaal als 1:25.000 tot en met zelfs kleine schalen (< 1:100.000).

Het blijkt dat met de opkomst van internet-GIS de behoefte aan on-the-fly generaliseren sterk is toegenomen. Ten eerste omdat de bandbreedte beperkt is en het dus van belang is dat er zo weinig mogelijk onnodige details worden overgestuurd; dat scheelt immers tijd. Belangrijk is dat snel een goed kaartbeeld wordt opgebouwd, dat eventueel later kan worden aangevuld met meer details, al dan niet na het inzoomen op een specifieke locatie. Ten tweede maakt internet-GIS het mogelijk om datasets van verschillende bronnen (en dus vaak ook van verschillende schalen) te combineren, maar dit is alleen zinnig indien eerst van alle datasets een presentatie op gelijke schaal wordt afgeleid.

Behalve de testresultaten en de implementatie van de standaard GAP-tree, wordt er een uitbreiding beschreven, die is gebaseerd op de toevoeging van parallelle lijnen aan de GAP-tree voor lineaire objecten zoals straten en waterwegen. Dit maakt het mogelijk om een betere en duidelijkere, in de breedte overdreven, representatie van deze objecten op kleinere schalen te tonen.

drs. J.D. van Putten[2]*), Nationaal Lucht- en Ruimtevaartlaboratorium te Marknesse, en prof. dr. ir. P.J.M. van Oosterom, TU Delft, afdeling Geodesie, sectie GISt.*

[1]) Gepresenteerd op het Internationaal Symposium over Spatial Data Handling, SDH'98 Vancouver, Canada, 12-15 juli 1998.

[2]) Dit onderzoek werd uitgevoerd tijdens de periode dat de auteur bij het Kadaster meewerkte aan dit project voor het behalen van haar doctoraalexamen.

Hoewel het gebruik van de GAP-tree zeer geschikt is voor efficiënte on-the-fly generalisatie, vergt het in één slag bouwen van de GAP-tree voor een grote dataset soms (te) veel computergeheugen. Daarom wordt er een nieuwe methode voorgesteld om van een grote dataset de GAP-tree representatie te bouwen, gebaseerd op een verdeel-en-heers techniek.

Dit artikel is opgesplitst in twee gedeelten. Dit eerste deel behandelt de theorie en de implementatie van de GAP-tree, terwijl het tweede gedeelte in het volgende nummer nader ingaat op de testresultaten en de verbeteringen.

Inleiding

Met de komst van Geografische Informatie Systemen (GIS) is de wijze waarop mensen kaarten of geografische data gebruiken drastisch veranderd. Moderne informatiesystemen maken het mogelijk door geografische datasets te navigeren door middel van het selecteren van (typen van) objecten en ze vervolgens op verschillende schalen af te beelden. Simpelweg verkleinen van objecten als de gebruiker uitzoomt, resulteert echter in een matige kwaliteit van de kaartrepresentatie. Bovendien wordt deze kaartrepresentatie erg langzaam opgebouwd, omdat deze veel te veel informatie bevat. Niet alleen moeten de objecten worden verkleind, maar ze moeten ook veel minder gedetailleerd worden weergegeven (vanwege de kleinere schaal). Minder belangrijke objecten moeten in hun geheel worden weggelaten, omdat ze voor de kleinschalige kaart niet van belang zijn. Een eenvoudige oplossing voor dit probleem is om de kaartrepresentatie op verschillende schalen, dus op verschillende detailniveaus, op te slaan. Dit brengt enkele serieuze nadelen met zich mee, zoals mogelijke inconsistenties tussen de verschillende representaties en een fors toegenomen gebruik van het ge-

heugen. Daarom zouden de geografische data moeten worden opgeslagen op een geïntegreerde manier zonder redundantie, en waar nodig, tevens ondersteund door een speciale datastructuur.

Detailniveaus zijn zeer nauw gerelateerd aan kartografische generalisatietechnieken. Het idee van on-the-fly kaartgeneralisatie [10] verschilt sterk van andere benaderingen, zoals beschreven in [5]. De term batch-generalisatie wordt gebruikt voor het proces waarin een computer een input dataset krijgt en een output dataset retourneert met gebruik van algoritmen, regels, of constraints [3] zonder menselijke tussenkomst. Dit in tegenstelling tot interactieve generalisatie waarin de gebruiker interactief met de computer werkt en een scala aan generalisatie-algoritmen kan toepassen en beoordelen voor gegeven situaties binnen de dataset. On-the-fly kaartgeneralisatie maakt geen tweede dataset, omdat dit redundante informatie zou opleveren. Het probeert een tijdelijke generalisatie te maken, alleen bedoeld om vluchtig op het scherm getoond te worden, op basis van een gedetailleerde geografische database. Snelle antwoorden, gevraagd door de interactieve gebruikers van een GIS, vereisen de toepassing van specifieke datastructuren, omdat anders de generalisatie te traag zou gaan voor datasets van enige omvang. Naast kaartgeneralisatie moeten deze datastructuren ook ruimtelijke selectie bieden, dat wil zeggen, het moet mogelijk zijn alle objecten binnen een specifiek gebied op efficiënte wijze te zoeken. Dit type datastructuren wordt 'reactive data structures' genoemd [7] [8] [9].

De GAP-tree, een reactieve datastructuur geschikt voor een vlakkenpartitie, wordt hierna kort beschreven en is gebaseerd op [10]. De implementatie van de GAP-tree wordt daarna in dit artikel behandeld. In het tweede artikel zal vervolgens worden gekeken naar de testresutaten, waarin de implementatie wordt geëvalueerd met behulp van twee datasets: vlakGBKN en Top10vector. Een aantal verbeteringen voor de GAP-tree worden in dit tweede artikel aangegeven. Tenslotte worden de conclusies besproken, samen met verwijzingen naar toekomstig werk.

De GAP-tree

Een vlakkenpartitie van de ruimte waarin elk punt in het 2D-domein tot precies één van de gebieden behoort, wordt vaak gebruikt als basisstructuur van de kaart, bijvoorbeeld topografische kaart, bodemkaart, kadastrale kaart en thematische choropleethkaart. Enkele problemen die ontstaan tijdens het toepassen van generalisatietechnieken zijn dan:

- *simplificatie.* Onafhankelijke lijngeneralisatie kan resulteren in lelijke kaarten omdat de vlakobjecten overlappingen en/of gaten kunnen bevatten. Een oplossing voor dit probleem is om een topologische datastructuur te gebruiken en lijngeneralisatie toe te passen op de gewone grenzen en randen (en niet direct op de vlakken zelf);
- *selectie.* Een onbelangrijk vlakobject weglaten zal een kaart produceren met een gat (*gap*) wat natuurlijk onacceptabel is. Tot voor de GAP-tree bestond nog geen duidelijke oplossing voor dit probleem.

Een gat dat ontstaat door het minst belangrijke vlakobject weg te laten, moet opnieuw worden ingevuld. De beste resultaten zullen worden verkregen door het gat met buurvlakken in te vullen. Dit kan vrij eenvoudig wanneer het gat een enclave of 'eiland' vormde in de topologische structuur; het kan worden opgevuld met het omliggende gebied. Bij 'niet-enclave' situaties zal een goede keuze een stuk moeilijker zijn.

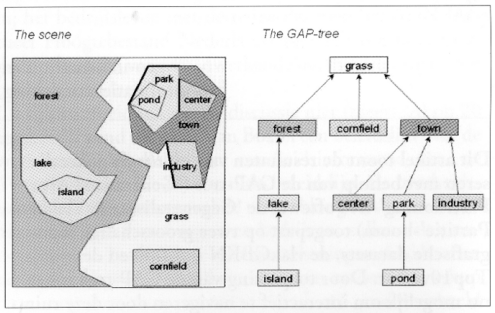

Fig. 1. Fragment van een topografische kaart met bijbehorende GAP-tree.

Het minst belangrijke vlakobject *a* heeft de laagste waarde voor de *Importantie* functie, bijvoorbeeld:

*Importantie(a) = prioriteit_t(a) * Gebied(a)*

waar *prioriteit_t(a)* de gewichtsfactor is voor de objectklassen (typen) waartoe vlakobject *a* behoort en *Gebied(a)* de oppervlakte is van vlakobject *a*. Buurman *b* met de hoogste waarde voor de *Samenvoeg(a,b)* functie mag het gat van vlakobject *a* invullen. Een voorbeeld *Samenvoeg(a,b)* functie is:

*Samenvoeg(a,b) = compatibiliteit_t(a,b) * Lengte(a,b)*

met *compatibiliteit_t(a,b),* de compatibiliteit tussen de objectklassen (typen) waartoe respectievelijk de vlakobjecten *a* en *b* behoren, en *Lengte(a,b)* de lengte van de gemeenschappelijke grens tussen *a* en *b*. De compatibiliteit is gerelateerd aan de semantische nabijheid van twee objectklassen in de objectclassificatie hiërarchie, die is geassocieerd met de betreffende dataset. Rekening houdend met de ruimtelijke relaties, de importantie van objecten en de compatibiliteit tussen objecten, wordt een hiërarchische vlakkenopdeling gecreëerd.

Een gewone planaire vlakkenopdeling van de ruimte wordt meestal opgeslagen in een topologische datastructuur met nodes (knopen), edges (randen) en faces (topologisch vlak). Het proces, hieronder beschreven, voor het maken van een hiërarchische vlakkenopdeling gaat uit van een topologische datastructuur [1] [2] [4] [6]. De topologische elementen bezitten de volgende attributen en relaties (hiervoor zijn verschillende implementaties denkbaar):

- een *node* (of 0-cel) bevat een punt en een lijst van referenties naar inkomende en uitgaande edges, gesorteerd voor de desbetreffende hoek;
- een *edge* (of 1-cel) bevat een polylijn (vorm), lengte, referenties naar de linker en rechter faces, en referenties naar de begin- en eindknopen;
- een *face* (of 2-cel) bevat de oppervlakte en een lijst van referenties naar randen die de buitenring vormen (een opeenvolging van randen die een gesloten keten vormen) en mogelijk binnenste ringen (gerelateerd aan eilanden). Deze referenties naar randen zijn elk van een + of – teken voorzien om de goede doorlooprichting van een rand aan te geven.

Deze topologische datastructuur bevat enige redundante informatie, die een efficiëntere verwerking in een later stadium mogelijk maakt. Nadat de topologische datastructuur is gemaakt, zal in de volgende stappen een hiërarchische vlakkenopdeling ontstaan, die in de zogenaamde GAP-tree wordt opgeslagen (meer details in [10]):

1. voor elke face (vlakobject) *a* in de topologische datastructuur wordt er een voorlopige 'losse' knoop in de GAP-tree gecreëerd en er wordt een toevoeging aan de prioriteitenrij gedaan, gebaseerd op de *Importantie(a)*-waarde;
2. selecteer het minst belangrijke vlakobject *a*. De prioriteitenrij ondersteunt dit via de operatie 'dequeue' om zo het vlakobject efficiënt op te zoeken;
3. gebruik de topologische datastructuur om de buren van *a* te vinden en bepaal voor elke buur *b*, de lijst van samenvallende gemeenschappelijke randen en de totale lengte hiervan *Lengte (a,b)*;
4. vul het gat in door buurman *b* met de hoogste functiewaarde *Samenvoeg(a,b)* te zoeken. Meer details met betrekking tot de stappen 3 en 4 worden in de volgende paragraaf behandeld;
5. sla de polygoon en andere attributen van vlakobject *a* in zijn knoop in de GAP-tree op en maak een tak in de boom van ouderknoop *b* naar kindknoop *a*;
6. voeg face *a* toe aan face *b*, pas de topologische datastructuur aan, herbereken de functiewaarde *Importantie(b)* en pas de positie van vlakobject *b* in de prioriteitenrij aan op basis van de nieuwe importantiewaarde.

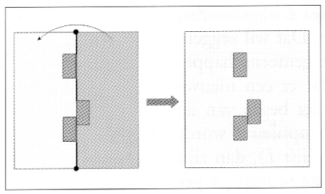

Fig. 2. Nieuwe enclaves kunnen ontstaan wanneer faces worden samengevoegd.

Herhaal de stappen 2-6 net zolang totdat er slechts één enorm vlakobject is overgebleven. Het resultaat is opgeslagen in de GAP-tree en het laatste vlakobject vormt de wortel (*root*) van de GAP-tree. Tijdens het interactieve gebruik in een GIS zal deze van tevoren berekende GAP-tree worden afgedaald totdat de knopen (behorende bij vlakobjecten) onder het vereiste importantieniveau voor de gewenste kaartrepresentatie dalen. Fig. 1 toont een fragment van een topografische kaart in de vorm van een vlakkenpartitie met daarnaast de GAP-tree, berekend volgens de hierboven beschreven methode. Elke polygoon staat op zichzelf en bevat alle coördinaten, en heeft dus geen lijst met verwijzingen naar randen, zoals de faces die wel hebben. De GAP-tree is geen binaire boom, maar kan per ouderknoop een willekeurig aantal kinderen hebben. Enkele visuele resultaten van de on-the-fly generalisatie met echte datasets zullen in het tweede artikel in het volgende nummer worden getoond.

Implementatie

Nu wordt meer in detail beschreven op welke wijze de beste buur voor samenvoegen kan worden gevonden, inclusief alle gemeenschappelijke randen met deze buur, en hoe de topologische datastructuur moet worden aangepast na het samengaan van twee vlakobjecten.

Hoe de beste buur te vinden (stappen 3 en 4)

De minst belangrijke face wordt gevonden via de prioriteitenrij in stap 2 en zal hier weer face *a* worden genoemd. De begrenzing van een face wordt gerepresenteerd via een lijst van verwijzingen naar randen die zowel de buiten- als de eventuele binnenringen (ingesloten enclaves) aangeven. Om een buur van face *a* te vinden, moet men naar een rand kijken en nagaan welke face er aan de andere kant ligt. Wanneer alle randen afgelopen zijn, zijn ook alle buurfaces bekend. Het is mogelijk dat een buur meerdere randen gemeenschappelijk heeft (fig. 2). Het is belangrijk dat al deze randen worden opgespoord nog voordat de *Samenvoeg(a,b)* functiewaarde wordt berekend, omdat deze is gebaseerd op de totale lengte van de gemeenschappelijke grens.

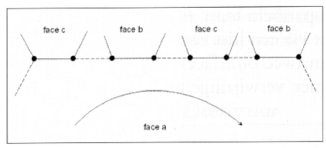

Fig. 3. Face a met mogelijke buren b en c.

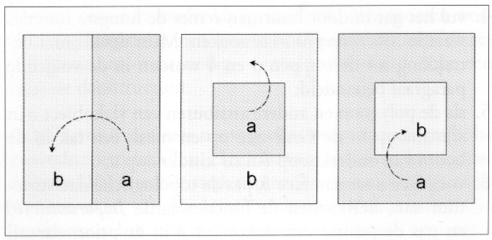

Fig. 4.
Drie situaties met van links naar rechts: Gewone buren (1), Toevoegen van enclave aan omhullende (1), Toevoegen omhullende aan enclave (1).

Gedurende het sequentieel doorlopen van de randen binnen een ring wordt er een, initieel lege, lijst *L* bijgehouden met daarin alle buurfaces in behandeling. Dat wil zeggen, de buren waarvan misschien nog niet alle gemeenschappelijke randen zijn gevonden. Elke keer dat er een nieuwe buur wordt gevonden, wordt deze aan het begin van de lijst *L* toegevoegd. Ingeval dat buur *b* opnieuw wordt gevonden (d.w.z. *b* stond al ergens in de lijst *L*), dan zijn alle buren in lijst *L* die voor buur *b* staan, compleet gevonden. Deze kunnen dan worden verwerkt en van lijst *L* worden afgevoerd. Op het moment dat alle randen van een face zijn doorlopen en er nog steeds buren in de lijst staan, kunnen deze ook worden verwerkt. Het verwerken houdt in: het berekenen van de *Samenvoeg(a,b)* functiewaarde en het bewaren van de buur met de hoogste functiewaarde tot nu toe. Merk op dat de beschreven procedure zowel voor de buitenring (face *a* is een normale buur of face *a* is een enclave binnen face *b*) alsook voor de binnenring werkt (face *a* omvat de enclave face *b*).

Hierboven werd gesteld dat, indien buur *b* opnieuw wordt aangedaan, alle buren voor buur *b* in de lijst *L* kunnen worden verwerkt. Het zal nu bewezen worden dat van deze buren alle gemeenschappelijke randen met face *a* inderdaad zijn gevonden via een bewijs uit het ongerijmde. Neem het tegenovergestelde aan, wat betekent dat er vóór buur *b*, een buur *c* in lijst *L* voorkomt, waarvan nog niet alle gemeenschappelijke randen gevonden zouden zijn. Hieruit volgt dan weer dat face *c* gemeenschappelijke randen heeft tussen en na de gemeenschappelijke randen die face *b* met face *a* heeft (fig. 3). Deze situatie impliceert echter dat de faces *b* en *c* elkaar kruisen, wat onmogelijk is in het geval van een planaire topologie. Indien de dataset correct is, kan dit niet voorkomen. Tegenspraak, einde bewijs.

Aanpassen topologische datastructuur (stap 6)

Deze paragraaf beschrijft hoe het minst belangrijke face *a* wordt samengevoegd met zijn meest compatibele buur *b*. Faces worden in de topologische structuur via een lijst van verwijzingen naar randen voorgesteld. Om twee buurfaces te vereningen, moeten de twee lijsten met verwijzingen naar randen (randlijsten) worden verbonden, de gemeenschappelijke randen worden verwijderd en moeten nieuwe enclaves die mogelijk ontstaan, worden ontdekt. Het hangt van de situatie af welke randlijsten met elkaar verbonden moeten worden. Er kunnen drie verschillende situaties mogelijk zijn, die in fig. 4 (eenvoudige gevallen) en in fig. 5 (complexere gevallen) worden getoond. Deze verschillende situaties kunnen met behulp van een polygoon-in-polygoon test (of punt-in-polygoon test met behulp van de centroïde) worden bepaald.

In de situatie 'normale buren' moeten de randlijst van de buitenring van face *a* en de randlijst van de buitenring van face *b* worden samengevoegd. Dit wil zeggen dat de gemeenschappelijke

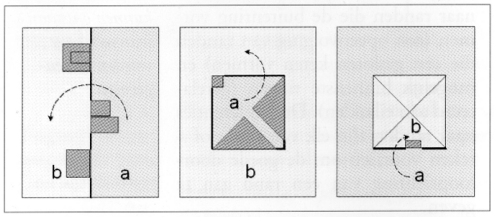

Fig. 5.
Drie situaties met van links naar rechts: Gewone buren (2), Toevoegen enclave aan omhullende (2), Toevoegen omhullende aan enclave (2).

randen worden verwijderd en de overgebleven delen van de randlijsten worden gekoppeld. In de situatie 'voeg enclave aan omhullende toe' moeten de randlijst van de buitenring van face *a* en de randlijst van de betreffende binnenring van face *b* worden samengevoegd. Het kan zijn dat hierdoor alle verwijzingen naar randen vervallen (fig. 4). In de situatie 'voeg omhullende aan enclave toe' moeten de randlijst van de betreffende binnenring van face *a* en de randlijst van de buitenring van face *b* worden samengevoegd. Merk op dat ook hier complete randlijsten kunnen wegvallen. De manier waarop de overgebleven delen van de randlijsten moeten worden samengevoegd, is in alle gevallen hetzelfde. Een gemeenschappelijke rand maakt deel uit van beide randlijsten. De randen vóór en na deze verwijderde gemeenschappelijke randen moeten nu met elkaar worden verbonden (fig. 6).

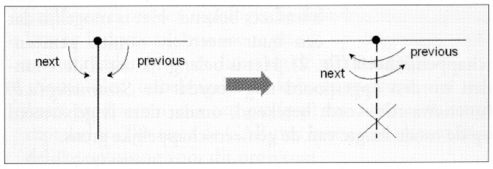

Fig. 6.
Verbinding van randlijsten.

Het verbinden van randlijsten moet bij alle verwijderde gemeenschappe-

lijke randen gebeuren, zowel bij de knopen vóór als na deze verwijderde rand. Tijdens dit proces van verbinden ontstaan weer nieuwe randlijsten die ringen vormen. Een nieuwe randlijst kan worden gevonden nadat een verwijderde gemeenschappelijke rand is verwerkt. De lijst met gemeenschappelijke randen *EL* kan voorwaarts of achterwaarts doorlopen en worden afgehandeld, maar de volgorde waarin die gemeenschappelijke randen worden afgehandeld, mag niet veranderen. In het geval dat een gemeenschappelijke rand *e* niet de eerste rand in de lijst *EL* is, zoek dan een nieuwe randlijst aan de kant van de knoop van de gemeenschappelijke rand *e* aan de kant waar de andere gemeenschappelijke randen al verwerkt zijn; bijvoorbeeld knoop *a* van rand *e2* in fig. 7, aannemende dat rand *e1* als eerste is verwerkt. Ingeval dat de gemeenschappelijke rand *e* de laatste rand in de lijst *EL* is (bijvoorbeeld *e3*), moet ook een nieuwe randlijst aan de andere kant van deze gemeenschappelijke rand worden gevonden.

Twee verschillende soorten randlijsten, die de nieuwe ringen van de samengestelde face representeren, kunnen nu zijn ontstaan. Dit zijn randlijsten met een draairichting met de klok mee en randlijsten die tegen de klok indraaien. Er zal precies één randlijst ontstaan, die met de klok meedraait; dit is de buitenring van de nieuwe face. De overige randlijsten, indien aanwezig, zullen alle tegen de klok indraaien en komen overeen met de binnenringen van de nieuwe face.

Bij het samenvoegen van twee faces is het mogelijk dat nieuwe 'loshangende' randen ontstaan, die verwijderd moeten worden. Dat wil zeggen, twee opeenvolgende verwijzingen naar randen in de nieuw gecreëerde randlijst wijzen naar dezelfde rand, maar wel met een tegenovergesteld teken. Dit zal zowel bij rand *e1* als rand *e2* in fig. 8 gebeuren. Deze randen moeten worden ontdekt en verwijderd. Merk op dat een nieuwe loshangende rand alleen kan ontstaan indien de oorspronkelijke rand zowel links als rechts dezelfde face had. Deze randen zijn niet nodig voor het werken met enclaves in een planaire partitie, maar kunnen in de dataset voorkomen om een lijnvormig object te representeren, bijvoorbeeld een hek. In plaats van het verwijderen van deze randen bij het bouwen van de GAP-tree, hadden ze ook vooraf verwijderd kunnen worden.

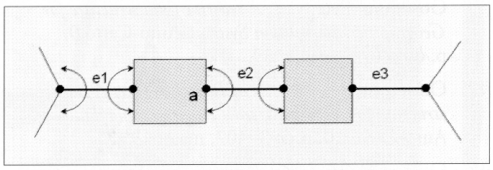

Fig. 7. Creatie van nieuwe (binnen)ringen.

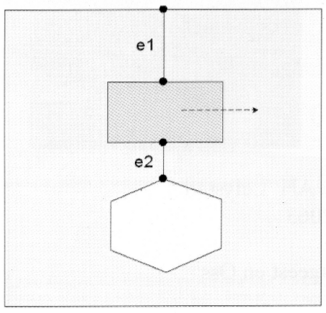

Fig. 8. Nieuwe loshangende randen.

Afsluiting

Deel twee van dit artikel gaat nader in op de resultaten van het testen van de GAP-tree bij het gebruik van een vlakGBKN en de Top10vector. Ook zullen twee belangrijke verbeteringen worden beschreven, namelijk parallelle lijnen om lineaire objecten beter te kunnen meenemen in het generalisatieproces, onder andere door deze te overdrijven, en een aanpassing om met zeer grote datasets om te kunnen gaan.

Summary

Generalization of Area Partitions (part 1)

The GAP-tree is a structure which can store generalizations in such a way that for every scale the matching generalization can be determined rapidly. In this research GAP-trees were built for two different datasets, the vectorGBKN and the Top10-vector. After setting the feature type priority and compatibility values, the generalizations were performed automatically. This is of great importance for efficient use of cartographic data via the internet. During webbrowsing, a correct generalization for every maplevel is determined and can be shown very fast. In this first part of the article the authors discuss the theoretical backgrounds and the implementation. In the next issue of this magazine the second part will deal with the test results and discuss possible improvements.

Literatuur

[1] Boudriault, Gerard, *Topology in the TIGER file.* Auto-Carto 8, p. 258-269, 1987.

[2] *DGIWG. DIGEST - digital geographic information - exchange standards - edition 1.1.* Technical report, Defence Mapping Agency, USA, Digital Geographic Information Working Group, oktober 1992.

[3] Lagrange, J. P., A. Ruas, L. Bender, *Survey on generalization.* Technical report, IGN, april 1993.

[4] Molenaar, Martien, *Single valued vector maps: A concept in Geographic Information Systems.* Geo-Informationssysteme, 2(1):18-26, 1989.

[5] Muller, J. C., R. Weibel, J. P. Lagrange, F. Salge, *Generalization: state of the art and issues.* Technical report, European Science Foundation, GISDATA Task Force on Generalization, 1993.

[6] Peucker, Thomas K. and Nicholas Chrisman, *Cartographic datastructures.* American Cartographer, 2(1):55-69, 1975.

[7] Oosterom, Peter van, *A reactive data structure for Geographic Information Systems.* Auto-Carto 9, p. 665-674, april 1989.

[8] Oosterom, Peter van, *The Reactive-Tree: A storage structure for a seamless, scaleless geographic database.* Auto-Carto 10, p. 393-407, maart 1991.

[9] Oosterom, Peter van, *Reactive Data Structures for Geographic Information Systems.* Oxford University Press, Oxford 1994.

[10] Oosterom, Peter van, *The gap-tree, an approach to 'on-the-fly' map generalization of an area partitioning.* In J.C. Müller, J.P. Lagrange, and R. Weibel, editors, GIS and Generalization, Methodology and Practice, p. 120-132. Taylor & Francis, 1995.