Het onderhouden van connectiviteit in ruimtelijke gegevens

Stephen MARTENS

promotor :
Prof. dr. Jan VAN DEN BUSSCHE

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica afstudeerrichting databases

universiteit hasselt

Abstract

Deze masterproef heeft als doel een FOIES (first order incremental evaluation system) te construeren voor de topologische connectiviteit van ruimtelijke gegevens.
De verschillende domeinen die nodig zijn voor het onderzoek worden eerst uitvoerig besproken en later gecombineerd om zo te komen tot het feitelijke onderzoek. Relationele databases en eerste orde logica worden gedefinieerd en besproken. Het principe van FOIES wordt uitvoerig uitgelegd en precies gedefinieerd. De constructie van een FOIES voor de transitieve sluiting van ongerichte grafen door Patnaïk en Immerman (die werkt m.b.v. een spanning forest) wordt volledig uit de doeken gedaan. De tekortkomingen van de constructie worden weggewerkt en aangevuld en correctheid van de uiteindelijke constructie zijn wordt aangetoond. Vervolgens wordt er een licht geworpen op constraint databases: ook hier worden de basisdefinities vastgelegd en de grondbeginselen goed uitgelegd. De belangrijke natural active collapse stelling wordt besproken en het principe van de implementatie van constraint databases wordt uitgelegd. Tenslotte wordt er ook nog het begrip van topologische connectiviteit gedefinieerd alvorens er overgegaan wordt tot het zoeken naar een FOIES. Hier wordt tevens ineens aangetoond dat topologische connectiviteit niet rechtstreeks uitdrukbaar is in FO.

Nadat deze domeinen uitvoerig besproken zijn, wordt er gefocust op het zoeken naar een FOIES om topologische connectiviteit incrementeel en decrementeel te berekenen. Er wordt op basis van de constructie van Patnaik en Immerman een FOIES geconstrueerd die de topologische connectiviteit van lijnstukken uitdrukt. De constructie is volledig FO incrementeel definieerbaar, maar is decrementeel niet sluitend. Er wordt een korte automatisatie ingevoerd die a.d.h.v. het aantal snijdende lijnstukken (met het te verwijderen lijnstuk) een formule selecteert die de update dan wel in FO kan volbrengen. Doch deze constructie is geen volledige FOIES. Tenslotte wordt er geëindigd met de vaststelling dat de constructie voor lijnstukken enkel gebruik maakt van de snijdende eigenschap, waardoor het vermoeden ontstaat dat de constructie gemakkelijk uitbreidbaar is naar parametriseerbare krommen.

Voorwoord

Na drie jaar universitaire studies weet je dan dat het moment aangebroken is om aan die gigantische klif te beginnen: het thesisjaar. Op het einde van de rit gekomen, krijgt voldoening de bovenhand, maar ook het besef dat dit karwei nooit tot een goed einde was gekomen zonder de hulp van vele mensen uit je naaste omgeving.

Dit voorwoord is dan ook de ideale gelegenheid om enkele mensen van harte te bedanken voor hun steun of begeleiding die ze mij geboden hebben het afgelopen jaar:

Prof. dr. Jan Van den Bussche, mijn promotor die zeer regelmatig nuttige brainstormsessies met mij heeft gehouden. Hij wist ook fouten snel te ontdekken waardoor er mij een groot tijdverlies bespaard bleef.

Dhr. Wim Janssen voor het nalezen van mijn thesis en voor zijn nuttige feedback.

Mijn ouders, voor de kansen die ze mij de afgelopen vier jaar hebben gegeven. Studeren is niet iedereen gegeven en kan enkel mogelijk gemaakt worden met de volledige steun van de ouders. Mijn moeder verdient ook een extra pluim voor het verbeteren van de spellingsfouten in mijn teksten.

Mijn collega-studenten, die voor de perfecte ontspanning zorgden tussen de lessen en het thesiswerk door.

Mijn handballende ploegmaats, die dit jaar dankzij sportieve successen voor een uitstekende uitlaatklep zorgden om de schoolstress kwijt te spelen.

Tot slot hoop ik dat deze masterproef met plezier gelezen mag worden en dat ze haar lezers iets kan bijbrengen.

Stephen Martens

Inhoudsopgave

Inleiding

Ruimtelijke gegevensbanken slaan gegevens op met een meetkundige interpretatie en hebben tal van toepassingen, b.v., GIS (Geographic Information Systems), CAD/CAM (Computer Aided Design and Manufacturing), en robotica. In deze toepassingen kan het belangrijk zijn om te weten of twee punten in een meetkundige wereld verbonden (*geconnecteerd)* zijn met elkaar.
Twee punten zijn verbonden wanneer zij beiden tot dezelfde connectiviteitscomponent behoren. Dit is het geval wanneer men van het ene punt naar het andere kan gaan zonder de figuur te verlaten.

Het onderzoeken of twee punten geconnecteerd met elkaar zijn is een zeer zware operatie voor ruimtelijke gegevensbanken. Alle figuren, behorend tot de meetkundige wereld, zullen onderzocht moeten worden om te zien of er ergens een verbinding kan gevonden worden om beide punten te connecteren. Voor vele applicaties is dit geen optie omdat er vaak een snelle responsetijd wordt verwacht. De connectiviteitsgegevens van de meetkundige wereld zullen dus op voorhand berekend en bijgehouden moeten worden. Het probleem is nu dat deze gegevens up-to-date moeten gehouden worden na updates. Dit probleem past in het ruimere kader van "view maintenance" (het onderhoud van views) in gegevensbanksystemen.

Deze masterproef zal een uiteenzetting geven van hoe deze gegevens op een efficiënte manier bijgehouden kunnen worden. Er zal gezocht worden naar een methode om de connectiviteitsgegevens op een snelle manier bij te werken wanneer er een update van de meetkundige wereld plaatsvindt.
De oplossing voor het probleem zal doorheen deze masterproef gegeven worden voor een meetkundige wereld die enkel lijnstukken bevat. Uiteindelijk zal deze oplossing geëvalueerd worden. Er zal onderzocht worden wat de oplossing aankan en waar zijn eventuele beperkingen liggen.

Hoofdstuk 1

Relationele Databases

Vooraleer wetenschappers bepaalde onderzoeken kunnen opstarten, moeten zij hun onderzoeksdomein duidelijk afbakenen. De hulpmiddelen, die in aanmerking komen tot gebruik, moeten duidelijk omschreven en eveneens goed afgelijnd worden.
In een theoretische studie moet net dezelfde voorbereiding getroffen worden waarbij zelfs de afbakening van het domein nauwkeuriger moet gebeuren.
In de eerste vier hoofdstukken van deze masterproef zullen verschillende domeinen gedefinieerd worden en belangrijke stellingen - nodig voor het eigenlijke onderzoek - zullen bewezen worden. De domeinen zullen uiteindelijk gecombineerd worden en vervolgens afgebakend worden voor het onderzoek.
In dit eerste hoofdstuk zal er een inleiding gegeven worden op relationele databases. Deze steunen op het relationele datamodel en worden traditioneel ondervraagd met behulp van de relationele algebra. Dit relationeel datamodel is bij informatici zeer bekend en zal daarom ook maar kort beschreven worden. In plaats van de relationele algebra zal het voor ons onderzoek beter zijn om gebruik te kunnen maken van een alternatieve taal: de eerste orde logica.

1.1 Het Relationeel Datamodel

Databases, zoals ze in het algemeen gebruikt worden, zijn gebaseerd op het relationeel datamodel. Dit model geeft een eenvoudige methode om een eindige hoeveelheid data voor te stellen. Deze data worden voorgesteld met behulp van (eindige) relaties. Relaties kunnen gezien worden als 2-dimensionale tabellen. We zullen de begrippen die we gaan invoeren telkens verduidelijken met het voorbeeld in fig. 1. De naam van de relatie is hier *Films* en is bedoeld om informatie bij te houden over bepaalde entiteiten van films.

In een relatie kunnen we *attributen* zien. Attributen dienen gewoonlijk om de betekenis van bepaalde eigenschappen van de entiteiten te omschrijven. In ons voorbeeld vormen de kolommen de attributen van de relaties. De bovenste rij vormen de attribuutnamen. Voor de relatie *Films* hebben we dus de attributen *titel, jaar, lengte* en *categorie.*

Titel	*Jaar*	*Lengte*	*Categorie*
Star Wars	1977	124	Science-fiction
Mighty Ducks	1991	104	Sport
Wayne's World	1992	95	Comedy

fig. 1 - Een voorbeeld relatie "Films"

De naam van een relatie en zijn verzameling attributen worden samen het *schema* van de relatie genoemd. Het schema wordt voorgesteld door de naam van de relatie, gevolgd door een opsomming tussen haakjes van de attributen, gescheiden door een komma. Het schema voor het voorbeeld in fig. 1 zal bijgevolg volgende uitdrukking zijn: *Films(titel, jaar, lengte, categorie).*
De attributen in een verzameling zijn een verzameling en geen lijst (dus er is geen volgorde bepaald en er mogen geen dubbels in de namen voorkomen). Er moet echter wel steeds duidelijk gespecificeerd worden over welke attributen het gaat. Om dit niet steeds weer erbij te moeten vermelden kan men de volgorde van de attributen in het gegeven schema als standaard volgorde nemen. De verzameling schema's van relaties wordt een relationeel database schema genoemd.

De ingevulde waarden van een relatie vormen de eigenlijke informatie die men wenst bij te houden. Zij vormen de *tupels* van een relatie. Een tupel heeft een component voor elk attribuut van de relatie. In ons voorbeeld is elke rij (buiten de rij met de attribuutnamen) een tupel van de relatie.
De uitdrukking (Star Wars, 1977, 124, Science-fiction) is een tupel van de relatie *Films*.

De ingevulde waarden voor elk attribuut in elk tupel moeten uit een voorafbepaald domein komen; zo is dit in ons gebruikte voorbeeld: (string, integer, integer, string), maar niet enkel de traditionele domeinen komen hier voor in aanmerking. Later in deze thesis zullen we zien dat hier ook constraints (o.a. ongelijkheden) voor gebruikt kunnen worden, die zelfs volledige figuren in het vlak definiëren. Hierdoor ontstaat dus de mogelijkheid om een oneindig aantal

hoeveelheid data te beschrijven in onze database. We zullen hier echter later nog op terugkomen.

Een relatie zal meestal dynamisch zijn, de verzameling tupels zal dus veranderen doorheen de tijd. Dit kan gebeuren door het toevoegen, verwijderen of veranderen van tupels. Op verschillende tijdstippen kan een relatie dus verschillende "inhouden" hebben. We zullen zo een momentopname van een inhoud een *instantie van een relatie* noemen. Formeel wil dit dus zeggen dat een instantie van een relatie een verzameling tupels is voor die relatie.
In fig. 1 is er bijvoorbeeld een instantie van de relatie *Films* gegeven, maar in 1980 zou de relatie een andere instantie hebben gehad (nl. enkel het tupel met de film *Star Wars*) [GUW02].

Om later enkele aandachtspunten duidelijk te kunnen illustreren, zullen we nu enkele belangrijke eigenschappen van standaard relationele query talen uitleggen.

- Een *query* is een formule in een bepaalde logica. De input is een verzameling relaties, net zoals de output.
- *Closure.* Talen zoals relationele algebra of FO zijn *gesloten (closed).* Dit betekent dat wanneer er een input in een bepaalde vorm (relaties) wordt gegeven, de output dezelfde vorm heeft (ook een relatie, zij het een andere). Dit is een sleuteleigenschap van het relationele model. De closure-eigenschap kan heel nuttig uitgebuit worden: een query kan bijv. opgedeeld worden in kleinere delen om ze te optimaliseren. Elk deel van de query kan dan zelf als een relationele query worden behandeld. Een ander nuttig gevolg van de eigenschap is dat dezelfde taal gebruikt kan worden voor zowel het querieën als het definiëren van views op een database.
- *Efficiente evaluatie.* Queries kunnen efficiënt geëvalueerd worden. De complexiteit waarmee dit gebeurt wordt meestal uitgedrukt in functie van de grootte van de database. Het bestaan van indexen zal daardoor meestal ook veel invloed hebben op die complexiteit.
- *Calculus/Algebra.* Het bestaan van equivalente talen biedt ook mogelijkheden. De equivalentie tussen de calculus (of logica) en algebraïsche querytalen maakt het voor de gebruiker mogelijk om queries te schrijven in een taal zoals de relationele calculus of SQL, terwijl het systeem een procedurele taal, zoals de relationele algebra, kan gebruiken om de queries te evalueren.

Relaties worden traditioneel beschreven en ondervraagd met behulp van de relationele algebra. Voor ons onderzoek zal het echter handiger zijn om met een alternatieve, equivalente querytaal te werken: nl. de eerste orde logica.

1.2 Eerste Orde Logica (FO)

De eerste orde logica, ook wel de relationele calculus genaamd, wordt de sleuteltaal van ons onderzoek. De verschillende queries die onze updates zullen moeten voltooien, zullen namelijk in eerste orde logica (vanaf hier afgekort tot FO) moeten lopen.

FO is opgebouwd uit kwantoren ($\forall$, $\exists$), connectoren ($\vee$, $\wedge$, $\neg$), constanten, variabelen en relaties. Deze relaties kunnen we vergelijken met de schema's uit de relationele algebra. Elke relatie kan opgebouwd zijn uit meerdere attributen (in FO *argumenten* genoemd). Een FO-expressie kan beschouwd worden als een query op de relaties waarover de FO-expressie handelt. Een expressie kan de argumenten van de relaties specificeren met behulp van constanten of variabelen. Een *variabele* laat toe om een willekeurig object te benoemen en er eigenschappen over te formuleren.
Kwantoren behoren ook tot instrumenten waarover de FO beschikt. Deze kwantoren zijn ook gekend uit de wiskunde. Men heeft de *universele kwantor* $\forall$, die uitdrukt dat alle objecten aan een uitdrukking voldoen. Naast de universele bestaat de *existentiële kwantor* $\exists$, die uitdrukt dat er bepaalde objecten bestaan.
Kwantoren die enkel toegelaten zijn om over objecten te spreken (en niet over verzamelingen van objecten), zorgen ervoor dat men binnen het gebied van eerste orde logica blijft. Indien men kwantoren laat reiken over verzamelingen van objecten (of verzamelingen van verzamelingen), dan spreekt men van hogere orde logica.

Kwantoren reiken in FO dus over variabelen. Variabelen kunnen in expressies *vrij* of *gebonden* zijn aan kwantoren. We definiëren dit nu formeel:

Definitie 1.1 Het bereik van een kwantor $(\exists v_n)$ of $(\forall v_n)$ op een bepaalde plaats in een expressie ϕ definiëren we als de kleinste deel-expressie ψ van ϕ die begint met die kwantor. Visueel krijgen we dus de volgende opdeling in expressies (in het geval van de existentiële kwantor):

$$\underbrace{\ldots \underbrace{(\exists v_n)(\ldots)}_{\psi} \ldots}_{\phi}$$

We zeggen nu dat een variabele v_n *gebonden* is op een bepaalde plaats in de formule ϕ als v_n op die plaats voorkomt binnen het bereik van een kwantor $(\exists v_n)$ of $(\forall v_n)$. In het andere geval is v_n *vrij* op die plaats. Een expressie waarin geen vrije variabelen voorkomen, noemen we een *zin* [KUI02].

We zullen vanaf hier een afkorting invoeren om verschillende variabelen aan een existentiële resp. universele kwantor te binden: in plaats van voor elke variabele de kwantor te herhalen, zullen we de variabelen groeperen en scheiden door een komma. We krijgen dus: $(\exists v_1,v_2,v_3)$ in plaats van: $(\exists v_1)(\exists v_2)(\exists v_3)$ en analoog voor de universele: $(\forall v_1,v_2,v_3)$ in plaats van: $(\forall v_1)(\forall v_2)(\forall v_3)$. Deze afkorting zal bij veel gebonden variabelen de leesbaarheid aanzienlijk verhogen.

Expressies zijn opgebouwd uit atomen. Vooraleer we dus expressies kunnen definiëren, zullen we moeten bepalen wat atomen precies zijn.

Definitie 1.2 De *atomen* van expressies ψ kunnen voorkomen in drie types van vormen:

1. $R(x)$, waarbij R een relatienaam is en x is een tupel variabele. Deze atoom staat voor de bewering dat x een tupel is in relatie R.
2. $x\ \theta\ y$, waarbij x en y variabelen zijn en θ is een rekenkundige vergelijkingsoperator (zoals <, =, …). Dit atoom staat voor de bewering dat de variabele x staat in relatie θ tot variabele y. Bijvoorbeeld: $R(x,y) \wedge x=y$ betekent dat relatie R bestaat uit twee variabelen (x en y), en elk tupel in de relatie zal twee gelijke waarden bevatten.
3. $x\ \theta$ c^{te}, waarbij x een single variabele is en de tweede parameter een constante. De betekenis is hier uiteraard gewoon analoog met het vorige type. Bijv: $R(x) \wedge x > 0$

betekent dat de relatie R enkel tupels bevat met strikt positieve getallen als waarde voor zijn enige variabele.

Nu het bereik van kwantoren en de atomen bepaald zijn, kan er overgegaan worden tot het definiëren van FO-expressies:

Definitie 1.3 Expressies, vrije en gebonden voorkomens van variabelen in deze expressies worden als volgt recursief gedefinieerd:

1. Elk atoom is een expressie. Alle voorkomens van tupelvariabelen in de atoom zijn *vrij* in deze expressie.
2. Als ψ_1 en ψ_2 expressies zijn, dan zijn $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ en $\neg\psi_1$ ook expressies. Deze expressies beweren respectievelijk dat *"zowel ψ_1 als ψ_2 waar zijn"*, dat *"ψ_1 of ψ_2 waar is (of eventueel ook beiden)"* en dat *"ψ_1 niet waar is."* De voorkomens van tupelvariabelen zijn *vrij* of *gebonden* afhankelijk van het feit of zij in de deelexpressie ψ_1 of ψ_2, waar zij in voorkwamen, ook *vrij* of *gebonden* waren. We merken op dat een voorkomen van variabele x in ψ_1 gebonden kan zijn, terwijl een ander voorkomen van x in ψ_2 vrij kan zijn (of omgekeerd).
3. Als ψ een expressie is, dan is $(\exists x)(\psi)$ ook een expressie. Het symbool $\exists$ is een kwantor. Naast $\exists$ is er nog een andere kwantor: nl. $\forall$ die in (4) besproken zal worden. $\exists$ wordt gelezen als *"Er bestaat…"* Alle voorkomens van x die in de expressie ψ *vrij* waren, zijn nu in de expressie $(\exists x)(\psi)$ *gebonden*. Andere voorkomens van variabelen (dus verschillend aan x) die *vrij* of *gebonden* waren in ψ, blijven ook nu resp. *vrij* of *gebonden*. De expressie $(\exists x)(\psi)$ beweert dat er een waarde van x bestaat, zodat de expressie ψ *waar* wordt, wanneer we alle vrije voorkomens van x in ψ vervangen door deze waarde. Bijvoorbeeld: $(\exists s)(R(s))$ zegt dat relatie R niet leeg is; m.a.w. er bestaat een tupel s in relatie R.
4. Als ψ een expressie is, dan is ook $(\forall x)(\psi)$ een expressie. De vrije voorkomens van x in ψ zullen nu echter gebonden zijn. De voorkomens van andere variabelen blijven *vrij* of *gebonden* naargelang ze *vrij* of *gebonden* waren in ψ. De expressie $(\forall x)(\psi)$ beweert dat welke mogelijke waarde ook wordt ingevuld voor de vrije voorkomens van x in ψ, de expressie zal altijd *waar* zijn.

5. Haakjes mogen rond expressies geplaatst worden wanneer dit nodig is. De prioriteitsregels die hier omtrent gevolgd worden zijn: *haakjes* hebben de hoogste prioriteit, daarna volgen de rekenkundige operatoren (<,=,…); vervolgens krijgen de kwantoren $\exists$ en $\forall$ voorrang op $\neg$, $\wedge$ en $\vee$ (in deze volgorde).
6. Niets anders kan als expressie worden beschouwd.

Definitie 1.4 Een *tupel eerste orde logica uitdrukking* is een uitdrukking van de vorm $R(t) \equiv \psi(t)$ waarbij t de enige vrije tupel variabelen zijn in ψ.
Voorbeeld. De unie U van relaties R en S wordt uitgedrukt door de FO-uitdrukking: $U(t) \equiv R(t) \vee S(t)$. We kunnen dus zeggen: U is de relatie die tupels bevat die tot R behoren of tot S. We merken op dat deze unie enkel zin heeft wanneer R en S evenveel argumenten hebben.

Bepaalde uitdrukkingen zullen vaak terugkomen in FO. We zullen daarom extra operatoren invoeren die de eerste orde logica niet uitbreiden, maar die simpelweg een verkorting zijn van veel terugkerende expressies; zo kunnen we het verschil tussen twee relaties R $\setminus$ S uitdrukken door: $R(t) \wedge \neg S(t)$ [Ull82].

Een andere nieuwe operator die we invoeren is $\rightarrow$. In woorden kan deze operator beschreven worden door *"als… dan…"*. De uitdrukking $R \rightarrow S$ is een afkorting voor $\neg R(t) \vee S(u)$ [KUI02].

1.3 Transitieve sluiting van een graaf

Er is dus een taal (nl. FO) opgesteld waarmee we onze database kunnen bevragen; toch kan FO niet alle mogelijke queries uitdrukken. Zo zal de transitieve sluiting van een graaf niet uitdrukbaar zijn in FO. In paragraaf §4.2 zal bewezen worden dat ook topologische connectiviteit niet rechtstreeks uitdrukbaar is in FO. Met "rechtstreeks" bedoelen we dat de query in één ruk het probleem oplost zonder gebruik te maken van tussenberekeningen bij bijvoorbeeld elke update. We zullen later in deze masterproef zien dat er wel meerdere problemen een oplossing kunnen hebben wanneer zij bij elke update de opgeslagen tussenoplossing aanpassen. Rechtstreekse queries zullen de oplossing recht-

streeks kunnen berekenen zonder tussenoplossingen te moeten opslaan.

Dat de transitieve sluiting van een binaire relatie niet rechtstreeks uitdrukbaar is in FO, is één van de eerst ontdekte voorbeelden van database queries die niet uitdrukbaar zijn in eerste orde logica. Vooraleer we dit kunnen bewijzen, zullen we de transitieve sluiting formeel definiëren:

Definitie 1.5 Zij R een binaire relatie. De transitieve sluiting van R is de kleinste transitieve relatie (in de zin dat $(a,b) \in R$ en $(b,c) \in R$ impliceert $(a,c) \in$ R en (b,c) impliceert $(a,c) \in R$) die R bevat.

We kunnen een graaf definiëren in FO met behulp van een binaire relatie $E(x,y)$ waarbij de argumenten de knopen x en y voorstellen en de relatie $E(x,y)$ de boog tussen de knopen. De definitie van transitieve sluiting kan dus ook geïnterpreteerd worden als de transitieve sluiting van een graaf.

In de praktijk komt het berekenen van de transitieve sluiting vaak voor. Bijvoorbeeld bij een luchtvaartmaatschappij: er kan een netwerk opgesteld worden met een binaire relatie die alle vluchten voorstelt; men kan vliegen van luchthaven x naar luchthaven y, dan stelt de transitieve sluiting alle paden voor die in dat netwerk gevolgd kunnen worden, (ongeacht het aantal overstappen dat gemaakt zou moeten worden).

In de cursus *Fundamenten van Databases* werd er formeel bewezen via Ehrenfeucht-Fraïssé-spellen dat de transitieve sluiting van een query niet uitdrukbaar is in de eerste orde logica [GV05].

Hoofdstuk 2

First Order Incremental Evaluation Systems

Het updaten van onze connectiviteitsgegevens willen we doen met behulp van eerste orde incrementele evaluatie (in de literatuur afgekort met FOIES = first order incrementel evaluation system).

De connectiviteitsgegevens berekenen in Eerste Orde (FO) logica is niet mogelijk (zie Hoofdstuk 4). Wat wel binnen onze mogelijkheden ligt, is om de connectiviteitsgegevens incrementeel te berekenen en deze voortdurend bij te houden. Concreet wil dit zeggen dat er vertrokken wordt van een lege wereld, waarin we stap voor stap meetkundige figuren gaan toevoegen of verwijderen. Per toevoeging of verwijdering zal nu de gekende informatie over connectiviteit worden geüpdatet. We zullen in deze masterproef proberen aan te tonen dat deze update wel in eerste orde logica kan gebeuren.

Deze incrementele techniek voor toevoegingen of verwijderingen staat bekend als FOIES (First Order Incremental Evaluation Systems).

2.1 Basisbegrippen

Wanneer we een probleem met FOIES willen oplossen, dan moet er aangetoond worden dat dit probleem *first order incrementeel definieerbaar* is voor toevoegen en/of verwijderen. Dus de relevante informatie (verder de oplossingsrelatie genoemd) zal bij elke toevoeging of verwijdering in FO moeten geüpdatet worden. We zullen eerst bepalen wat we verstaan onder een toevoeging of een verwijdering:

Definitie 2.1 *Een* toevoeging *(resp.* verwijdering*) over een binair predikaat R is een mapping* Δ^+ *(resp.* Δ^-*) van inst(R)* $\rightarrow$ *inst(R), met* $\Delta \in$ *inst(R) als singletonverzameling, gedefinieerd als* $\Delta^+(r) = r \cup \Delta$ *(resp.* $\Delta^-(r) = r \setminus \Delta$*).*

Wanneer er dus *k* figuren willen toegevoegd (of verwijderd) worden, dan kan dit gezien worden als een opeenvolging van *k* toevoegingen (of verwijderingen) zoals hierboven gedefinieerd. Nu deze begrippen duidelijk bepaald zijn, kan er overgegaan worden tot de notie van FO incrementele en decrementele definieerbaarheid. Er zijn twee soorten van zulke definieerbaarheid: nl. één is waar een probleem kan gedefinieerd worden zonder hulprelatie, het tweede is waar er beroep moet worden gedaan op een hulprelatie. Voor elk probleem zal er constant de *oplossingsrelatie* worden bijgehouden. Deze oplossingsrelatie is de relatie die bij elke toevoeging/verwijdering geüpdatet moet worden met behulp van een eerste orde logica query.

Definitie 2.2 *Stel dat R een binair predikaat is en* $C \subseteq$ *inst(R). Een probleem P over R wordt* FO incrementeel definieerbaar *over C genoemd als er een mapping Q bestaat (uitdrukbaar in eerste orde logica) die als input een instantie van R, de bijgehouden oplossingsrelatie O en een toevoeging krijgt, zodat voor elke* $r \in C$ *en voor elke toevoeging* Δ *(met* $r \cup \Delta \in C$*), de nieuwe oplossingsverzameling* $O_{r \cup \Delta} = Q(r, O_r, \Delta)$*.*
Een probleem P over R wordt FO incrementeel definieerbaar *over C* met een hulprelatie *genoemd als er een predikaat R', een mappende functie f:* $C \rightarrow$ *inst(R') en twee mappings Q en Q' (beiden uitdrukbaar in FO) bestaan met als input een instantie van R' (de hulprelatie), de huidige oplossingsverzameling O en een toevoeging zodat voor elke* $r \in C$ *en voor elke toevoeging* Δ *(met* $r \cup \Delta \in C$*) geldt dat:*

(1) de nieuwe oplossingsverzameling is $O_{r \cup C} = Q(r, O_r, f(r), \Delta)$
(2) de overeenstemmende nieuwe hulprelatie is:
$f(r \cup \Delta) = Q'(r, O_r, f(r), \Delta)$*.*

Voor het verwijderen verloopt de eigenschap FO decrementele definieerbaarheid volledig analoog waarbij enkel de toevoegingen $r \cup \Delta$ vervangen moeten worden door de verwijderingen $r \setminus \Delta$ [DS95].¨

2.2 Incrementele definieerbaarheid van de transitieve sluiting van grafen

Indien enkel toevoegingen worden beschouwd, dan kan er voor alle grafen bewezen worden dat de query die de transitieve sluiting berekent, incrementeel definieerbaar is.
De transitieve sluiting van een graaf bevat alle koppels van knopen waarbij de eerste knoop van elk koppel het tweede kan bereiken via een pad binnen de graaf.

Definitie 2.3 *De* query voor de transitieve sluiting *van een graaf G over een binair predikaat R is een mapping TC van inst(R) naar inst(R) gedefinieerd als TC(r) = {(x,y) | er bestaat een pad van knoop x naar knoop y in graaf* G_r*} voor elke* $r \in inst(R)$*}.*

We zullen nu bewijzen dat we de transitieve sluiting van een gegeven graaf G_r incrementeel definieerbaar is; d.w.z. dat we vertrekken van een lege graaf (en een lege transitieve sluiting) en dat we bij het toevoegen van een nieuwe boog aan de graaf, dat de transitieve sluiting dan geüpdatet kan worden, met behulp van een eerste orde logica query.

Stelling De transitieve sluiting query bij toevoegingen in grafen is eerste orde incrementeel definieerbaar.

Bewijs. Beschouw een graaf G_r met zijn transitieve sluiting TC_r. De binaire relatie R is de relationele representatie van G_r:

$$\forall\, x,y\text{: boog}(x,y) \in G_r \Leftrightarrow R(x,y)$$

Stel: Δ is een toevoeging van boog(a,b) aan graaf G_r.
Dus: $G_{r \cup \Delta} = G_r \cup \{ (a,b) \}$

Dan is de nieuwe transitieve sluiting:

$$TC_{r \cup \Delta} = \{ (x,y) \mid TC_r(x,y) \vee (TC_r(x,a) \wedge TC_r(b,y)) \vee (x{=}a \wedge y = b) \}$$

Deze FO expressie is een disjunctie van 3 deelexpressies, die we als volgt benoemen:

$\phi_1 = TC_r(x,y)$
$\phi_2 = (TC_r(x,a) \wedge TC_r(b,y))$
$\phi_3 = (x{=}a \wedge y = b)$

De expressie $\phi_1 \vee \phi_2 \vee \phi_3$ behoort tot FO volgens de definitie van FO (Definitie 1.3).

Bijgevolg zien we duidelijk dat als deze constructie correct is, dat de transitieve sluiting per toevoeging van een boog wordt geüpdatet met behulp van een query in eerste orde logica.
Er moet dus nog aangetoond worden dat deze constructie correct is:

We bewijzen dit door aan te tonen dat bovenstaande FO-query de geüpdatete transitieve sluiting van de graaf uitdrukt ($\Rightarrow$) en de andere richting is dat een boog toevoegen aan de graaf overeenkomt met de bovenstaande FO-query ($\Leftarrow$).

Basis: Voor n = 0:
($\Rightarrow$) We hebben de lege relatie TC_R. De graaf bevat nog geen bogen. Het is dus triviaal dat ook de transitieve sluiting nog geen enkel koppel knopen zal bevatten. Bij toevoeging van een boog (a,b) zal dus $\phi_1 \vee \phi_2$ al zeker geen tupels teruggeven (want TC is nog leeg). Enkel ϕ_3 zal er voor zorgen dat de nieuwe transitieve sluiting nu enkel het koppel (a,b) bevat. Dit is correct want door toevoeging van de eerste boog aan een graaf, kan de transitieve sluiting enkel de knopen bevatten die de boog verbindt (en wel nog in die volgorde).

($\Leftarrow$) De graaf bevat nog geen bogen. We vertrekken dus van een lege transitieve sluiting. Bij toevoeging van een boog (a,b), zal de nieuwe transitieve sluiting van de graaf één koppel bevatten; nl. het koppel (a,b). In de relatie TC_R bevat na toevoeging van (a,b) de geüpdatet relatie ook enkel dit koppel. Onze basisstap is dus in orde.

Inductiehypothese:
($\Rightarrow$) Wanneer aan de binaire relatie R (met *n-2* tupels) een koppel Δ wordt toegevoegd, dan drukt de nieuwe relatie $TC_{R \cup \Delta}$ de nieuwe transitieve sluiting van de graaf G_R(met *n-1* bogen) uit:
$TC_{R \cup \Delta} = \{ (x,y) \mid TC_r(x,y) \vee (TC_r(x,a) \wedge TC_r(b,y)) \vee (x=a \wedge y = b) \}$

($\Leftarrow$) Wanneer aan de graaf G_R (met n-2 bogen) een boog Δ wordt toegevoegd, dan wordt de nieuwe transitieve sluiting van de graaf (met *n-1* bogen) uitgedrukt door de relatie:
$TC_{R \cup \Delta} = \{ (x,y) \mid TC_r(x,y) \vee (TC_r(x,a) \wedge TC_r(b,y)) \vee (x=a \wedge y = b) \}$.

Inductiestap:
(⇒) Wanneer aan de binaire relatie R (die *n-1* tupels bevat) een koppel (*a*,*b*) wordt toegevoegd en de relatie TC_R wordt vervolgens geüpdatet dan kunnen de tupels van de nieuwe relatie $TC_{R\cup\{(a,b)\}}$ worden opgedeeld in 3 groepen (overeenkomstig met de 3 deelexpressies van de disjunctie):

1. De tupels die voldoen aan ϕ_1 zijn de tupels die reeds in de oude relatie zaten. Dit zijn dus de paden van de graaf die reeds bestonden voordat de nieuwe boog werd toegevoegd.
2. Met ϕ_2 krijgen we de tupels die een verbinding maken tussen alle koppels (x,a) en alle koppels (b,y), die in de oude relatie zaten. Deze nieuwe koppels vormen de nieuwe paden die kunnen gevormd worden in de graaf door alle bestaande paden aankomende in *a* te verbinden met alle bestaande paden vertrekkende in *b*.

Er komt één tupel overeen met ϕ_3: nl. (a,b). Dit komt overeen met de nieuwe boog (en dus ook het nieuwe pad) van *a* naar *b*.

(⇐) Wanneer een boog (*a*,*b*) wordt toegevoegd aan de graaf G_R (met n-1 bogen), dan kunnen er in de nieuwe transitieve sluiting twee gevallen optreden:

1. Een pad van knoop *x* naar knoop *y* in de graaf, gaat niet langs (*a*,*b*). In dit geval zou dat pad reeds in de transitieve sluiting gezeten hebben voor de graaf met n-1 bogen. Dit is het geval dat wordt uitgedrukt door ϕ_1.
2. Een pad van knoop *x* naar knoop *y* in de graaf, gaat wel langs (*a*,*b*). Door toevoeging van de nieuwe boog zijn er nieuwe paden ontstaan. Een nieuw pad kennen we al zeker: nl. van knoop *a* naar knoop *b*. Dit nieuwe pad wordt uitgedrukt door de FO-query met deelexpressie ϕ_3. De andere nieuwe paden ontstaan uit de oude transitieve sluiting. Elk pad dat in de oude vertrekt in knoop *x* en aankomt in knoop *a* en elk pad dat in de oude vertrekt in knoop *b* en aankomt in knoop *y*, zal nu een nieuw pad vormen in de nieuwe transitieve sluiting van knoop *x* naar knoop *y;* dit is precies wat deelexpressie ϕ_2 uitdrukt. We kunnen dus besluiten dat bij toevoeging van een nieuwe boog, de nieuwe transitieve sluiting juist wordt berekend door de genoemde eerste orde logica query. □

2.3 Een FOIES voor de transitieve sluiting van ongerichte grafen

Binnen de graaftheorie weten we nu dat de transitieve sluiting (TC) incrementeel berekend kan worden in FO voor elke graaf. Het is echter nog steeds een open probleem of de TC ook decrementeel (stap per stap verwijderen van één boog) bijgehouden kan worden in FO.
Voor enkele speciale vormen van grafen zijn er wel al constructies gevonden, waarbij de TC ook decrementeel berekend kan worden in FO. Zo hebben Dong en Su dit aangetoond voor acyclische grafen en voor 0-1 pad grafen (grafen waarbij elk paar knopen met hoogstens één pad verbonden zijn met elkaar) [DS95]. Patnaik en Immerman zijn met een FOIES gekomen voor ongerichte grafen. Om een FOIES te ontwikkelen voor topologische connectiviteit gaan we ons baseren op het principe van de constructie van Patnaik en Immerman. In deze paragraaf zullen we de constructie uitvoerig bespreken en zijn correctheid aantonen.

2.3.1 Spanning Forest

We gaan het begrip uit de graaftheorie van spanning forests invoeren. Dit zal gebruikt worden in de FOIES constructie voor de transitieve sluiting van ongerichte grafen.

Definitie 2.4 Een graaf G is opgebouwd uit verschillende samenhang-componenten. Voor elke samenhangcomponent kan een *spanning tree* opgebouwd worden. Als de component bestaat uit k knopen en m bogen, dan zal zijn spanning tree een deelgraaf zijn met alle k knopen die elk verbonden worden door $k-1$ bogen. De spanning tree mag *geen* cycli bevatten. Alle spanning trees van de graaf samen vormen het *spanning forest* van de graaf [Big74].

Voorbeeld. In fig. 2 wordt er een graaf als voorbeeld gegeven waarbij ook de bogen gelabeld zijn. In de graaftheorie kan er namelijk zelfs m.b.v. een algoritme de minimum spanning tree worden gezocht (minimum omdat die bogen gekozen worden waarvan de som van hun labels het kleinste is). In fig. 3 wordt zo een minimum spanning tree gegeven, wij zullen echter enkel kijken naar de typische eigenschappen van de spanning tree.

Op de figuur kan men heel duidelijk zien dat de spanning tree acyclisch is (terwijl de originele graaf dit helemaal niet was). De originele graaf was één grote samenhangcomponent, dus alle knopen van de graaf moeten in de spanning tree voorkomen. Er zijn 10 knopen die in de spanning tree verbonden zijn door 9 bogen.

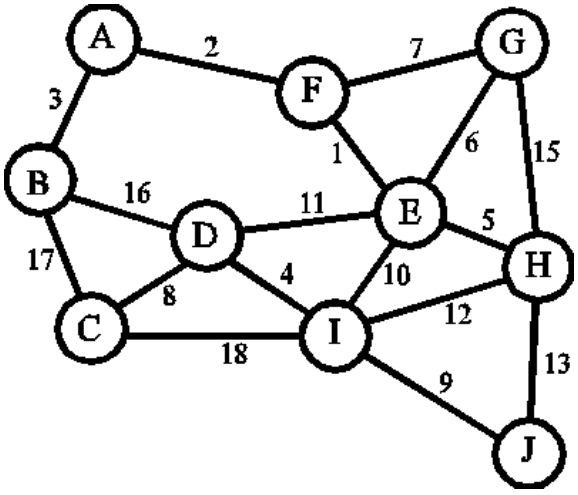


fig. 2 - Voorbeeld van een gewogen graaf

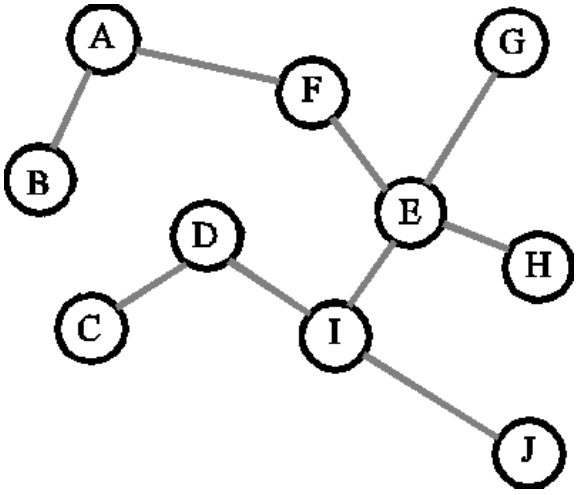


fig. 3 - Minimum spanning tree van de graaf

2.3.2 FOIES constructie

De grafen die we gaan bestuderen zijn allemaal ongerichte grafen. We gaan een spanning forest bijhouden voor de graaf met behulp van volgende relaties:

- $\hat{F}(x,y)$ is de relatie die de bogen (x,y) bevat van de huidige spanning tree. Deze relatie zal steeds symmetrisch zijn (want we werken met ongerichte grafen). We zullen daarom een afkorting invoeren waarbij we de ongerichtheid steeds rechtstreeks mee uitdrukken:
$F(x,y) \equiv \hat{F}(x,y) \vee \hat{F}(y,x)$.
- $\hat{PV}(x,y,u)$ betekent dat er een (uniek) pad bestaat in het forest van knoop x naar knoop y via knoop u. De knoop u kan ook één van de eindpunten zijn (dus x of y). Net zoals bij de vorige relatie zullen we hier ook de symmetrische eigenschap van de ongerichte grafen afkorten tot één relatie: $PV(x,y,u) \equiv \hat{PV}(x,y,u) \vee \hat{PV}(y,x,u)$.
Ter illustratie: als $F(x,y)$ waar is, dan zijn ook $PV(x,y,x)$ en $PV(x,y,y)$ waar.
- $P(x,y)$ is een afkorting om aan te duiden of er een pad in de graaf bestaat van knoop x naar knoop y. Dit kunnen we zien door te kijken of er een onderling pad bestaat in het forest tussen beide knopen. Het is ook mogelijk dat x en y dezelfde knoop voorstellen. Dit brengt ons tot volgende afkorting: $P(x,y) \equiv (x = y) \vee PV(x,y,x)$.
We merken ook op dat deze relatie onze uiteindelijke oplossing van ons probleem zal uitdrukken. De constructie wordt tenslotte opgesteld om op een in- of decrementele manier de transitieve sluiting bij te houden.

Ook bij het invoegen en verwijderen zullen we de ongerichtheid interpreteren door de operatie op zowel boog (a,b) als op (b,a).

Insert(E,a,b). Voor de eenvoud zullen we nu de update van de relaties E, F en PV beschrijven door F' en PV' i.p.v. door $E_{r\cup\{(a,b)\}}$, $F_{r\cup\{(a,b),(b,a)\}}$ en $PV_{r\cup\{(a,b)\}}$.
Het onderhouden van de boogrelatie E is triviaal:
$E'(x,y) \equiv E(x,y) \vee (x=a \wedge y=b) \vee (x=b \wedge y=a)$.
We zullen het effect van het toevoegen van boog (a,b) bespreken.
Als knopen a en b reeds in dezelfde samenhangende component zaten, dan blijven de bogen in het forest onveranderd. Indien a en b elk tot een andere component behoorden, dan zal er in het spanning forest een boog bijkomen van a naar b, zodat de twee samenhangende compo-

nenten waartoe zij behoren samengevoegd worden tot één samenhangende component [fig. 4]. Of er twee knopen tot dezelfde component behoren, is hetzelfde als kijken of er een pad bestaat tussen beide knopen:

$\hat{F}'(x,y) \equiv \hat{F}(x,y) \vee (\neg P(a,b) \wedge ((x=a \wedge y=b) \vee (x=b \wedge y=a)))$.

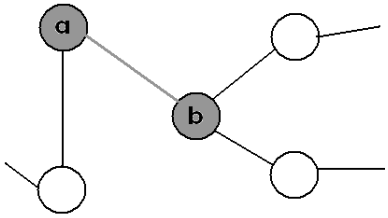


fig. 4 - Knoop a en b behorend tot een verschillende component en beide componenten worden verbonden door de nieuwe (a,b)

Vervolgens moet relatie *PV* geüpdatet worden. Net zoals de bogen van het forest ontstaan er enkel nieuwe paden binnen het forest wanneer *(a,b)* ervoor zorgt dat twee verschillende samenhangcomponenten worden samengevoegd. In dit geval moeten alle knopen van de samenhangcomponent waartoe *a* behoort gelinkt worden aan alle knopen van de component waartoe *b* behoort.
Het komt er dus op neer om alle paden van de ene component die eindigen in knoop *a* te verlengen met alle paden van de andere component die vertrekken in knoop *b*. Al die paden (en de knopen *u* waarlangs ze lopen) worden dus gewoon telkens overgenomen en de knopen *a* en *b* worden er nog aan toegevoegd (want daarlangs passeren de nieuwe paden).

$$\hat{PV}'(x,y,u) \equiv \hat{PV}(x,y,u) \vee [P(x,\mathrm{a}) \wedge P(\mathrm{b},y) \wedge \neg P(\mathrm{a},\mathrm{b}) \wedge (PV(x,\mathrm{a},z) \vee PV(\mathrm{b},y,z) \vee z=\mathrm{a} \vee z=\mathrm{b})\ ].$$

Met behulp van fig. 5 kunnen we de opgestelde expressie van PV beter begrijpen.
Het eerste deel van de expressie, nl. *PV(x,y,u)* is zeer eenvoudig te begrijpen. Het is de overname van de bestaande relatie PV. Het tweede deel van de expressie bevat de eigenlijke update:
De deelexpressie $P(x,a) \wedge P(b,y)$ duidt aan dat de knopen *x* uit de component van *a* komen en de knopen *y* uit de component waarin ook knoop *b* zit. De overblijvende deelexpressie specificeert voor elk pad langs welke knopen het pad liep (de expressie specificeert dus de 3e parameter *u*). De nieuwe paden zijn paden uit de ene component

gelinkt aan de paden van de andere component via de boog *(a,b)*. Een pad loopt dus langs een knoop *z* waarvan we weten dat die *ofwel* behoorde tot het pad van de component van *a* (geval z_1 op de figuur), *ofwel* tot het pad van de component van *b* (geval z_2 op de figuur).

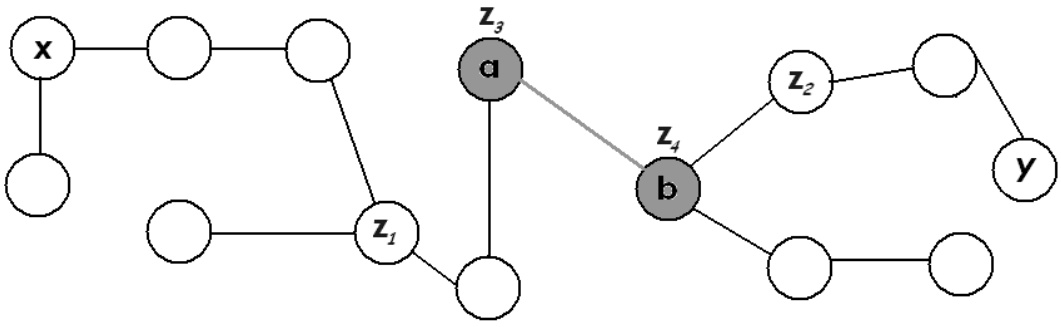


fig. 5 - Alle paden in de spanning tree van de component van a moeten met alle paden in de spanning tree van de component van b gelinkt worden

Het geval z_1 wordt uitgedrukt door *PV(x,a,z)*; dus het pad van knoop *x* naar knoop *a* dat langs knoop *z* loopt. We hebben een analoog geval voor z_2 en deelexpressie *PV(b,y,z)*. De laatste deelexpressie $z=a \vee z=b$ drukt uit dat het nieuwe pad gebruik maakt van boog *(a,b)*; m.a.w. het nieuwe pad loopt langs knoop *a* en langs knoop *b*. Al deze nieuwe paden ontstaan enkel indien a en b nog niet tot dezelfde component behoorden, wat dus getest wordt met de laatste conjunctie: $\neg P$(a,b*)*.

Bijkomende opmerking. De laatste deelexpressie $z=a \vee z=b$ is een expressie die ik zelf heb toegevoegd aan de constructie omdat Patnaik en Immerman hier niet volledig in waren. In hun paper gebruikten ze dezelfde expressie zonder $z=a \vee z=b$, dit is echter onvolledig want vertrekkend van een lege relatie zou deze relatie nooit geüpdatet worden omdat de deelexpressie $PV(x,a,z) \vee PV(b,y,z)$ steeds leeg zou zijn. Ook wanneer de relatie niet meer leeg is, is de constructie onvolledig want het nieuwe pad van knoop *a* naar knoop *b* (wat ervoor nog niet bestond indien *a* en *b* tot een verschillende component behoorden) zou met hun constructie niet opgenomen worden in de relatie *PV*'. Door deze aanvulling was het enorm belangrijk dat (a,b) nog niet in de spanning tree zat, daarom heb ik ook deze voorwaarde ($\neg P$(a,b)) aan de conjunctie van die deelexpressie moeten toevoegen.

delete(E,a,b). Net zoals bij het invoegen van een nieuwe boog zullen bij het verwijderen van een boog uit de graaf, de relaties *F* en *PV* in eerste orde logica moeten geüpdatet worden. Dankzij de opbouw van het

spanning forest is het nu mogelijk om deze relaties up-to-date te houden voor ongerichte grafen. Voor grafen in het algemeen is het nog niet geweten of dit mogelijk is, maar hier komen we later nog op terug.

In eerste instantie kunnen we al zeggen dat als boog *(a,b)* verwijderd wordt, maar diezelfde boog zit niet in onze forest (en *F(a,b)* is dus *false),* dan hoeven we niets te veranderen aan onze relaties, behalve gewoon de boog *(a,b)* uit de boogrelatie E verwijderen.
Dit klopt want indien boog *(a,b)* niet in het spanning forest zit, dan kan dit alleen maar betekenen dat knopen *a* en *b* elkaar kunnen bereiken via een ander pad in de spanning tree dat geen gebruik maakt van de verwijderde boog *(a,b),* want boog *(a,b)* zat niet in het forest en *a* en *b* behoren wel tot dezelfde component.

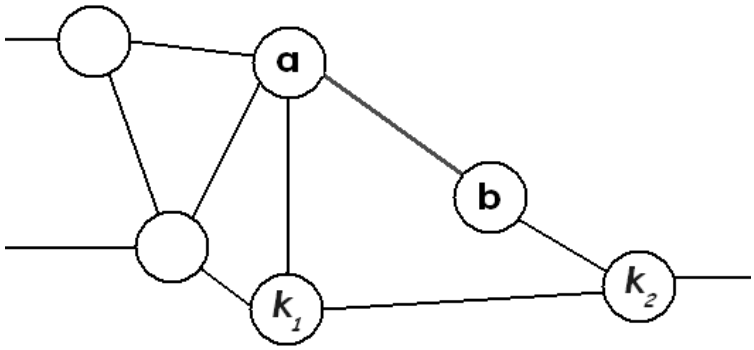


fig. 6 - Het verwijderen van boog (a,b) uit de graaf

Als we fig. 6 bekijken zijn er twee mogelijkheden.
Ten eerste: in het opgebouwde spanning forest zit boog *(a,b)* niet, in dat geval wil dat zeggen dat knopen *a* en *b* elkaar op een alternatieve manier kunnen bereiken. In dit geval via knopen k_1 en k_2. We besluiten dus dat de bogen (a,k_1), (k_1,k_2) en (b,k_2) wel tot de spanning tree behoren (zie fig. 7).

Het andere geval: wanneer boog *(a,b)* wel in de spanning tree zit, dan zullen de forest-relaties wel moeten geüpdatet worden.

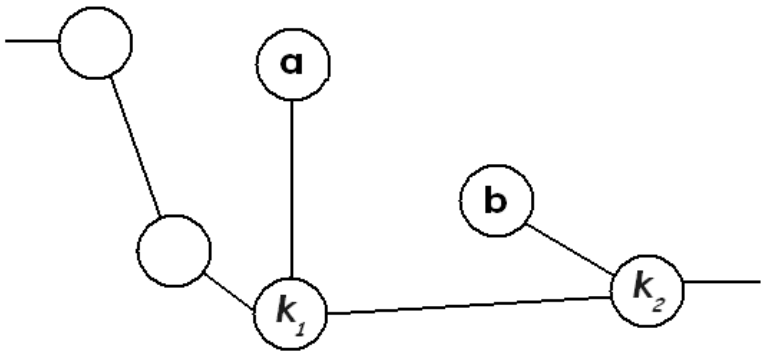


fig. 7 - Mogelijke spanning tree zonder boog (a,b)

Concreet zal er onderzocht moeten worden of (na het verwijderen van *(a,b)*) beide knopen nog behoren tot dezelfde samenhangcomponent; m.a.w. er zal onderzocht worden of het verwijderen van boog *(a,b)* de samenhangcomponent doet uiteenvallen in twee samenhangcomponenten (en dus in twee spanning trees). In onze constructie (zie ook fig. 8 ter illustratie) zorgt het verwijderen van de (bestaande) boog *(a,b)* in onze spanning tree ervoor dat de bestaande spanning tree uiteenvalt in twee verschillende spanning trees (in ons voorbeeld zijn dit de nieuwe trees F_a en F_b). Er zal onderzocht worden of er een boog bestaat in de graaf die ervoor zorgt dat a en b elkaar kunnen bereiken op een alternatieve manier. In het gegeven voorbeeld komt dit concreet neer op het vinden van de boog (k_1,k_2). Door het toevoegen van deze boog zou er een nieuw pad ontstaan in de spanning tree waarlangs a en b elkaar kunnen blijven bereiken (zoals ook mogelijk is in de originele graaf waarover het spanning forest is opgebouwd).

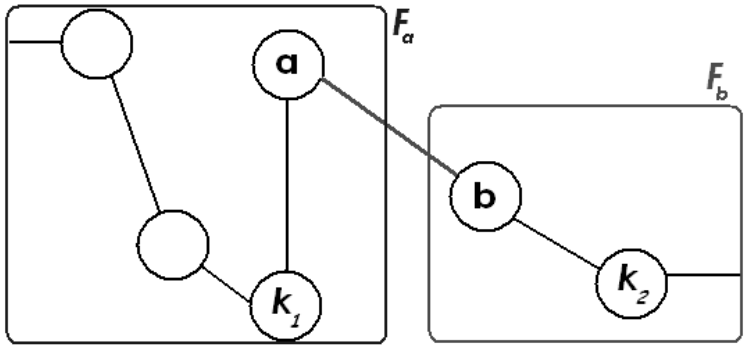


fig. 8 - Spanning tree die boog (a,b) wel bevat en waar het verwijderen van die boog de spanning tree doet breken in twee verschillende trees

We bekijken nu hoe onze FO-queries deze mogelijkheden zullen aanpakken. Allereerst kunnen we al zeggen dat de boogrelatie *E* eenvoudig geüpdatet kan worden:
$E'(x,y) \equiv E(x,y) \wedge \neg((x=a \wedge y=b) \vee (x=b \wedge y=a))$.

Voordat we verder gaan, gaan we enkele relaties definiëren die de opbouw van onze andere queries zullen vereenvoudigen:

1. *T(x,y,u)* is een tijdelijke relatie waarbij we de *PV* relatie kunnen uitdrukken waaruit de boog *(a,b)* is verwijderd (dus m.a.w. *T* is de relatie *PV* waaruit de paden zijn verwijderd die langs knoop *a* en langs knoop *b* passeerden:
   $T(x,y,u) \equiv PV(x,y,u) \wedge \neg(PV(x,y,a) \wedge PV(x,y,b))$.
2. *New(x,y)* maakt gebruik van de tijdelijk gedefinieerde relatie T om een eventuele boog te vinden die de uiteengevallen component toch terug verbindt. De relatie zal in de geüpdatete boogrelatie *E'* op zoek gaan naar een boog waarvan het ene eindpunt behoort tot de gebroken component F_a (waartoe knoop *a* behoort) en het andere eindpunt tot de andere gebroken component F_b (waar knoop *b* zich in bevindt). Indien hier meerdere bogen voor in aanmerking komen, dan nemen we de boog die de kleinste knopen verbindt(1).

   $$New(x,y) \equiv E'(x,y) \wedge T(a,x,a) \wedge T(b,y,b) \wedge (\forall u,v)\ [(\ E'(u,v) \wedge T(a,u,a) \wedge T(b,v,b)\ ) \rightarrow (x < u \vee (x=a \wedge y \leq v)\ )\ ].$$

   Het eerste gedeelte (nl. $E(x,y) \wedge T(a,x,a) \wedge T(b,y,b)$) van de FO expressie drukt uit dat knoop *x* behoort tot de component F_a en knoop *y* behoort tot F_b. Het tweede gedeelte drukt uit dat alle andere bogen *(u,v)* die gevonden worden en waarvan beide eindpunten ook tot de verschillende componenten F_a en F_b behoren, groter zijn dan de gekozen (minimale) boog *(x,y)*.

(1) We merken hierbij op dat er gebruik wordt gemaakt van een orderelatie op de knopen. Dit is steeds mogelijk. Indien er geen orde zou gegeven zijn op de knopen, dan kan men ze gewoon ordenen op volgorde van invoegen. Deze volgorde zou heel eenvoudig in een extra relatie kunnen bijgehouden worden.

De updates van de bogen in het forest zijn zeer eenvoudig. De boog *(a,b)* moet gewoon uit het forest worden verwijderd en de nieuwe boog moet (symmetrisch) worden toegevoegd:

$$\hat{F}'(x,y) \equiv [\ \hat{F}(x,y) \wedge \neg(\ (x=a \wedge y=b) \vee (x=b \wedge y=a)\ )\ ] \vee New(x,y) \vee New(y,x).$$

De paden in het forest, die niet langs de boog *(a,b)* passeerden, zijn nog steeds geldig. Dit zijn dus de paden die momenteel in de tijdelijke relatie *T* zitten. In het geval dat F_a en F_b nog steeds tot dezelfde samenhangcomponent behoren (ondanks dat boog *(a,b)* verwijderd werd uit de graaf), hebben we een nieuwe boog (k_1,k_2) die voor nieuwe paden zorgt. Deze nieuwe paden moeten dus ook toegevoegd worden aan de geüpdatete relatie *PV'*. Dit zijn alle paden die nog in de tijdelijke relatie zitten en eindigen in knoop k_1, verbonden met alle paden in de tijdelijke relatie *T* die vertrekken in knoop k_2 d.m.v. de aan het forest toegevoegde boog (k_1,k_2) (zie opnieuw fig. 8 ter verduidelijking).
De hiervoor opgestelde expressie verloopt een beetje analoog wat betreft het linken van paden via de nieuwe boog. Net zoals toen heb ik de constructie van Patnaik en Immerman moeten aanvullen om het FOIES volledig en juist te laten werken. Het verschil was nu echter dat enkel a en b moesten gespecificeerd worden en dat er geen extra conditie meer moest toegevoegd worden. Deze conditie zit nl. reeds vervat in de voorwaarde dat er een *New(x,y)* bestaat:

$$\hat{PV}'(x,y,z) \equiv T(x,y,z) \vee [\ (\exists\ k_1,k_2)\ (New(x,y) \vee New(y,x)) \wedge T(x,k_1,x) \wedge T(y,k_2,y) \wedge (T(x,k_1,z) \vee T(y,k_2,z) \vee z=a \vee z=b)\ ].$$

De analogie met de update van *PV* bij het toevoegen is gemakkelijk terug te vinden. Het enige verschil is dat we nu niet terugblikken naar onze vorige versie van *PV* maar naar de tijdelijk gedefinieerde relatie *T* (dus zonder de paden via de verwijderde boog *(a,b)*) en dat de link niet wordt gelegd via een gegeven boog *(a,b)* maar de link verloopt via een berekende boog *New(x,y)* [PI95].

We hebben nu de volledige constructie en werking uitgelegd van het ontwikkelde systeem van Patnaik en Immerman. In de volgende paragraaf zal aangetoond worden dat het systeem wel degelijk in FO loopt en dat het gewenste resultaat wel degelijk het correcte resultaat is.

2.3.3 Correctheid van de FOIES voor ongerichte grafen

Om aan te tonen dat de constructie die in vorige paragraaf werd uitgelegd wel degelijk werkt, moeten we zijn correctheid aantonen. In eerste instantie komt dit erop neer dat we moeten nagaan of alle updates wel degelijk in FO gebeuren. Vervolgens zal er moeten aangetoond worden dat het geconstrueerde spanning forest steeds uit verschillende spanning trees zal blijven bestaan.

We kunnen de gebruikte update-formules opsplitsen in 2 groepen.
De eerste groep formules zijn kwantorvrije formules die de nieuwe relatie updaten, vertrekkende van de oude (maar eigenlijk een andere) relatie en die met behulp van enkele disjuncts en conjuncts van relaties tupels bijvoegen of verwijderen.

Bijvoorbeeld: $\hat{PV}'(x,y,u) \equiv \hat{PV}'(x,y,u) \vee$
$[P(x,a) \wedge P(b,y) \wedge \neg P(\mathrm{a},\mathrm{b}) \wedge$
$(PV(x,a,u) \vee PV(b,y,u) \vee u=a \vee u=b)\ ]$.

We zien dus dat deze formule geen kwantoren bevat en dat de nieuwe relatie enkel gedefinieerd wordt door een disjunctie of conjunctie van relaties (of constraints zoals z=a $\vee$ z=b). Dit soort formules behoort zeker tot FO volgens Definitie 1.3: alle variabelen zijn vrij en komen terug in het queryhoofd. We merken op dat a en b hier geen variabelen zijn, maar constanten.

De andere soort formules zijn de niet-kwantorvrije formules. Deze zullen we één na één bespreken:

$\hat{PV}'(x,y,z) \equiv T(x,y,z) \vee$
$[\ (\exists\ k_1,k_2)\ (New(x,y) \vee New(y,x)) \wedge T(x,k_1,x) \wedge T(y,k_2,y) \wedge$
$(T(x,k_1,z) \vee T(y,k_2,z) \vee z=a \vee z=b)\ ]$.

In deze formule hebben we een kwantor $\exists$ over de variabelen k_1 en k_2, Deze variabelen zijn bijgevolg gebonden. De overige gebruikte variabelen (x,y,z) zijn vrij en die zien we dan ook terugkomen in het queryhoofd. Opnieuw merken we hier op dat a en b constanten zijn en geen variabelen.
Dit is analoog met de andere query met kwantoren:

$New(x,y) \equiv E'(x,y) \wedge T(a,x,a) \wedge T(b,y,b) \wedge$
$(\forall \mathrm{u},\mathrm{v})\ [(\ E'(u,v) \wedge T(a,u,a) \wedge T(b,v,b)\ )$
$\rightarrow (x < u \vee (x=a \wedge y \leq v)\ )\ ]$.

Voor beide queries kunnen we dus volgens Definitie 1.1 (bereik kwantoren) en volgens Definitie 1.3 (opbouw FO formules) besluiten dat elke update van de constructie verloopt in FO.

Een spanning forest bestaat uit alle spanning trees van de graaf in kwestie. De belangrijkste eigenschappen van een spanning tree is dat alle knopen van een samenhangcomponent behoren tot dezelfde spanning tree. Alle knopen zijn met elkaar verbonden via één uniek pad en er mogen geen cycli voorkomen in de spanning tree.

Voor de constructie is het belangrijk dat de spanning tree wel degelijk een spanning tree blijft, zodat de paden nog juist geüpdatet blijven.
We zullen nu aantonen dat in alle mogelijke gevallen de eigenschappen van de spanning tree blijven behouden.

Eerst zullen de updates van het invoegen beschouwd worden, later het verwijderen:

Invoegen van boog (a,b)

Het updaten van de forest-relatie zelf gebeurt met de formule:
$\hat{F}'(x,y) \equiv \hat{F}(x,y) \vee (\neg P(a,b) \wedge ((x=a \wedge y=b) \vee (x=b \wedge y=a)))$.

We vertrekken steeds met een geldige spanning tree:
Er zijn 2 mogelijkheden:

- In het spanning forest bestaat reeds een pad van a naar b (dus $\neg P(a,b)$ geeft *false*). In dit geval gebeurt er niets nieuws (want de deelformule is een conjunctie). Knoop a kon knoop b al bereiken (weliswaar via een langer pad), door het toevoegen van (a,b) ontstaan er nu dus geen nieuwe paden. Er komt niets bij, het is dus triviaal dat de spanning tree blijft voldoen. □
- Knoop a kon knoop b nog niet bereiken ($\neg P(a,b)$ geeft *true*). In dit geval wordt de ongerichte boog (a,b) toegevoegd aan de forestrelatie (d.m.v. de formule $(x=a \wedge y=b) \vee (x=b \wedge y=a)$. Beide knopen behoorden dus nog niet tot dezelfde spanning tree. Door een ongerichte boog tussen hen toe te voegen, is het dus ook onmogelijk om een cyclus te doen ontstaan. We bewijzen dit in het ongerijmde:
  Bewijs. We veronderstellen dat het toevoegen van boog (a,b) een cyclus in de spanning tree, waartoe a en b behoren, doet ontstaan. Dit wil automatisch zeggen dat de ontstane cyclus

(a,b) op zijn pad heeft liggen (want voordien hadden we nog een geldige (acyclische) spanning tree). Met als gevolg dat als we vertrekken in *a* en via boog *(a,b)* naar *b* gaan en de graaf verder volgen dat we via een bepaald pad (dat geen gebruik meer maakt van *(a,b)*) terug in *a* kunnen komen. Dit wil m.a.w. zeggen dat zonder boog *(a,b)* er reeds een ongericht pad bestond van knoop *b* naar knoop *a*. Dit is echter onmogelijk want we hebben gegeven dat *P(a,b) false* geeft (zie ook fig. 9). □

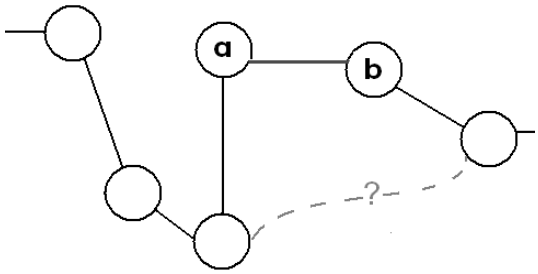


fig. 9 – Het toevoegen van boog (a,b) kan onmogelijk een lus creëren want dan zou P(a,b) reeds bestaan hebben (*waar* geweest zijn).

Verwijderen van boog (a,b).

Het updaten van het spanning forest gebeurt nu met volgende FO-formule:

We vertrekken ook hier steeds weer met een geldige spanning tree en net zoals bij het toevoegen kunnen we nu twee mogelijke gevallen onderscheiden:

- Boog *(a,b)* zit niet in het spanning forest. Dit wil zeggen dat we reeds een alternatief pad kennen in de spanning tree om van knoop *a* knoop *b* te bereiken (kijk terug naar fig. 7). Het verwijderen van *(a,b)* uit de graaf zal dus geen consequenties hebben, aan het spanning forest zal dus niets gewijzigd worden. Het is dus triviaal dat het nieuwe spanning forest nog steeds is opgebouwd uit geldige (en dezelfde) spanning trees.
- In het andere geval zit boog *(a,b)* wel in het spanning forest. Doordat *(a,b)* verwijderd wordt uit de graaf, zal deze boog ook verwijderd moeten worden uit het spanning forest. Als we *(a,b)* verwijderen uit het spanning forest, dan zal de spanning tree, waartoe *a* en *b* behoorden, uiteen vallen in twee verschillende componenten. De constructie gaat dan een nieuwe boog zoeken

die de twee componenten terug verbindt (indien die bestaat). We gaan nu onderzoeken of de boog die hiervoor gebruikt wordt geen lus veroorzaakt. Dit is echter triviaal bepaald. Aangezien de twee componenten uit elkaar vallen is er geen koppeling meer tussen beiden. De eerdere samenhangcomponent bevatte geen lussen, dus de twee uiteengevallen componenten zullen ook geen lussen bevatten. Wanneer beide componenten verbonden worden door één boog toe te voegen is het onmogelijk om een lus te creëren (fig. 10).

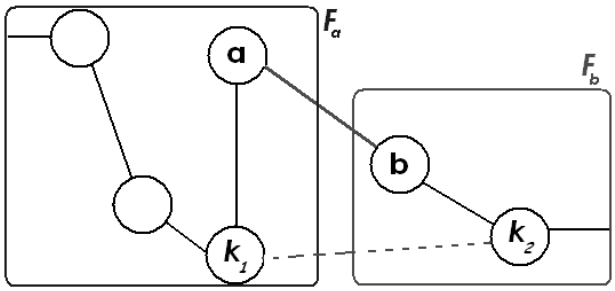


fig. 10 - Spanning tree valt uiteen in 2 componenten (door delete(a,b)) Door toevoeging van (k1,k2) is het nu onmogelijk een cyclus te creëren

We zullen dit nu nagaan aan de hand van de update-formules. De nieuwe boog wordt gezocht door de formule

$$New(x,y) \equiv E'(x,y) \wedge T(a,x,a) \wedge T(b,y,b) \wedge (\forall u,v)\,[(E'(u,v) \wedge T(a,u,a) \wedge T(b,v,b)) \rightarrow (x < u \vee (x=a \wedge y \leq v))].$$

Deze formule drukt uit dat er een boog [$E'(x,y)$] gezocht wordt waarvan de ene knoop behoort tot de component van knoop a [$T(a,x,a)$] en de andere tot die van b [$T(b,y,b)$]. Indien er zo meerdere bogen worden genomen, wordt de kleinste genomen $[(\forall u,v)\,[(E'(u,v) \wedge T(a,u,a) \wedge T(b,v,b)) \rightarrow (x < u \vee (x=a \wedge y \leq v))]$.

Dit laatste is belangrijk, want indien er meerdere bogen in aanmerking komen om de componenten terug te verbinden, dan mag er toch maar één boog terug aan de spanning tree worden toegevoegd. Indien er hier meerdere zouden toegevoegd worden zou de acyclische eigenschap van de spanning tree in gevaar komen. In onze afleiding die hierboven werd gegeven werd er ook duidelijk van uitgegaan dat er maximaal één boog werd toegevoegd, waardoor onze afleiding dus nog steeds correct is.

Het eigenlijke updaten van het spanning forest gebeurt met de formule:

$\hat{F}'(x,y) \equiv [\ \hat{F}(x,y) \wedge \neg(\ (x=a \wedge y=b) \vee (x=b \wedge y=a)\ )\ ] \vee New(x,y) \vee New(y,x).$

Zoals gezegd wordt hier gewoon de ongerichte boog *(a,b)* uit de spanning tree verwijderd [$\neg(\ (x=a \wedge y=b) \vee (x=b \wedge y=a)\ )$] en wordt de nieuwe ongerichte boog toegevoegd:
[$\vee\ New(x,y) \vee New(y,x)$].
Het enige wat we nog moeten staven is dat het verwijderen van de boog *(a,b)* de spanning tree wel degelijk doet uiteen vallen in twee samenhangcomponenten: dit volgt uit de definitie van spanning tree (Definitie 2.4). Aangezien een spanning tree alle *k* knopen van de component met elkaar verbindt met *k-1* bogen is het onmogelijk, wanneer een boog verwijderd wordt, dat dan nog steeds alle knopen op één of andere manier met elkaar verbonden zijn. Aangezien er nu nog maar *k-2* bogen zijn kunnen er nog slechts *k-1* knopen verbonden zijn met elkaar. De component zal dus steeds uiteenvallen in twee componenten (waarvan één component mogelijkerwijs slechts één knoop bevat, maar dit verandert niets aan onze afleiding). □

Het bewijs dat de samenhangcomponent bij het verwijderen wel degelijk in 2 componenten wordt gesplitst, wordt met een eenvoudig voorbeeldje duidelijk gemaakt in fig. 11.

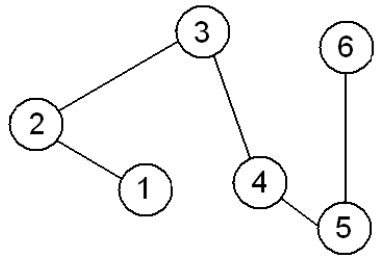


fig. 11 - Een spanning tree met 6 knopen en 5 bogen. Bij het verwijderen van één boog valt de tree onvermijdelijk in 2 componenten uiteen

De forest-relatie wordt enkel aangepast door de updates. We hebben aangetoond dat vertrekkende van een geldige spanning forest, de nieuwe relatie na de update nog steeds een geldig spanning forest zal zijn. Indien we vertrekken van een lege graaf en een lege spanning forest (wat uiteraard een geldig forest is), dan kan men door enkel met

updates bogen toe te voegen alleen maar een geldig spanning forest houden.
De enige aanname die tijdens onze afleiding nu nog gemaakt werd en die we nog niet aangetoond hebben is dat het vaststellen dat 2 knopen al dan niet tot één component behoren, altijd juist zal zijn. Met andere woorden: aantonen dat de *PV* relatie steeds correct wordt geüpdatet.

Bij het invoegen wordt de *PV* relatie als volgt geüpdatet:

$$\hat{PV}'(x,y,u) \equiv \hat{PV}'(x,y,u) \vee [P(x,a) \wedge P(b,y) \wedge \neg P(\text{a,b}) \wedge (PV(x,a,u) \vee PV(b,y,u) \vee u{=}a \vee u{=}b)\ ].$$

Om aan te tonen dat deze updates steeds een juiste weergave zijn, zullen we vertrekken van een lege verzameling die stap na stap wordt opgebouwd. Bij een lege verzameling ontstaan de 2 paden in de relatie van a naar b via resp. knoop a en knoop b. Deze stap is triviaal en de basisstap geeft dus een correcte weergave.

Als inductiehypothese stellen we dat na *n-1* updates de *PV* relatie nog steeds juist werd geüpdatet. We zullen nu onderzoeken of de n^e update correct verloopt. Bij het toevoegen van boog *(a,b)* zijn er net zoals bij het updaten van *F* opnieuw twee mogelijkheden die kunnen optreden:

- Knopen a en b zaten reeds in dezelfde spanning tree (fig. 7). In dit geval zal de deelformule $\neg P(\text{a,b})$ *false* geven en zullen er dus geen nieuwe paden toegevoegd moeten worden (aangezien er ook geen wijzigingen aan de spanning tree werden gebracht).
- Knopen a en b behoorden nog niet tot dezelfde spanning tree (fig. 5). Bij het updaten van *F* hebben we gezien dat de spanning trees waartoe a en b behoorden aan elkaar gekoppeld worden door het toevoegen van boog (a,b) aan het spanning forest. Bijgevolg ontstaan er nieuwe paden in de spanning tree. Deze nieuwe paden zijn de nieuwe tupels die de effectieve update van *PV* vormen. Er zijn dus enkel tupels wanneer a en b nog niet in dezelfde component zitten (en er dus nog geen pad bestond tussen hen). Daarom dat er enkel tupels ingevoegd worden wanneer $\neg P(\text{a,b})$ *true* geeft. De nieuwe paden die ontstaan zijn simpel te omschrijven. Het zijn alle paden van de ene component die overgenomen moeten worden en gekoppeld aan alle paden van de andere component, verbonden via boog (a,b). Deze tupels zijn op te delen in drie groepen afhankelijk van wat de derde parameter *z* zal zijn (dus langs welke knopen de paden lopen).

- Het nieuwe pad loopt langs een knoop z die op het pad lag van de eerste component (van a). Deze worden uitgedrukt door $PV(x,a,u)$.
- Dit geval is analoog met het vorige: nu komt echter het pad langs de knopen die behoren tot de paden van de spanning tree van de andere component: $PV(b,y,u)$
- De nieuwe paden lopen allemaal langs (a,b), wat aangeduid wordt door: z=a $\vee$ z=b.

We hebben alle deelformules ontleed en zijn nu dus zeker dat enkel nieuwe (en juiste) paden worden toegevoegd wanneer er ook daadwerkelijk iets verandert aan de spanning tree. We hebben ook de verschillende mogelijkheden behandeld waardoor we zeker zijn dat alle updates zijn uitgevoerd en dat er dus geen weggelaten zijn. We kunnen dus besluiten dat de update voor het invoeren correct gebeurd.

Om de correctheid van een update, waarbij er verwijderd wordt, aan te tonen, moeten we iets uitgebreider te werk gaan:

De update gebeurt in enkele stappen. Eerst zal de boog(a,b) verwijderd worden uit de spanning tree. Daarom wordt in eerste instantie een tijdelijke relatie T opgesteld die alle paden, die lopen langs de boog (a,b), uit de spanning tree verwijderd,. Hoe weten we dat een pad loopt via de boog(a,b)? Wel omdat het gaat om een uniek pad in de spanning tree en omdat de spanning tree acyclisch is, kan men eenvoudig de paden opvragen die zowel langs knoop a als langs knoop b gaan. De formule om de tijdelijke relatie T op te stellen is juist: $PV(x,y,u) \wedge \neg(PV(x,y,\text{a}) \wedge PV(x,y,\text{b})$.

De correctheid van het zoeken naar de nieuwe boog is reeds aangetoond. We kijken nu wanneer die nieuwe boog gevonden is of de PV relatie juist wordt geüpdatet. De update zelf gebeurt met de formule: $\hat{PV}'(x,y,z) \equiv T(x,y,z) \vee$

$$[(\exists\ k_1,k_2)\ (New(k_1,\ k_2) \vee New(k_2,\ k_1)) \wedge T(x,k_1,x) \wedge T(y,k_2,y) \wedge (T(x,k_1,z) \vee T(y,k_2,z) \vee z=a \vee z=b)]$$

Zoals vereist vertrekken we van de tijdelijk opgestelde relatie (waarin geen paden meer zitten die gebruik maken van boog (a,b)). Indien er geen nieuwe boog gevonden wordt, dan zal de nieuwe relatie gelijk zijn aan de tijdelijke relatie. Dit klopt want de conjunct ($New(x,y) \vee New(y,x)$) stellen dan lege relaties voor en geven altijd *false* waardoor

de hele conjunctie *false* wordt en waardoor dus enkel de tupels van *T(x,y,z)* in de nieuwe PV zullen zitten.
Indien er wel een boog gevonden wordt om de twee componenten te verbinden, dan wordt er volledig analoog gewerkt met het principe van het toevoegen van een ongerichte boog (a,b) met het verschil dat het nu gaat om het toevoegen van de ongerichte boog (k_1, k_2) die de *new* boog voorstelt, die de 2 componenten terug verbindt.
Het enige verschil in deze deelformule is dat de nieuwe boog niet gespecificeerd wordt met constanten a en b, maar dat deze nu bepaald worden met behulp van de existentiële kwantor die de knopen bepaalt met de correcte hulpquery *New*.
We mogen dus nu besluiten dat de PV relatie correct wordt bijgehouden (vertrekkende van een lege verzameling).

De FOIES die we geconstrueerd hebben werkt bijgevolg volledig correct. We besluiten dat de transitieve sluiting van ongerichte grafen (net zoals het bijhouden van de spanning tree voor ongerichte grafen) incrementeel uitdrukbaar is in FO. □

Hoofdstuk 3

Constraint Databases

In het vorige hoofdstuk hebben we gezien dat we met behulp van FO queries kunnen schrijven voor relationele databases.
In deze thesis willen we werken met ruimtelijke gegevens. Daarom zullen we nu kijken hoe we zulke gegevens kunnen stockeren in een database. Alle punten die behoren tot een figuur gaan stockeren, is onmogelijk want tot een eenvoudig lijnstuk behoren al een oneindig aantal punten. Daarom zullen we onze toevlucht nemen tot zo-genaamde constraints. Hierbij zullen we namelijk de mogelijkheid hebben om oneindige relaties uit te drukken.

3.1 Het principe

Een constraint (beperking) kunnen we intuïtief bekijken als een reeks voorwaarden die opgelegd worden om de verzameling punten (van de figuur die we met de constraint willen omschrijven) te bepalen ("beperken").
Met behulp van het constraint database model kan men een databank opbouwen waarbij de datawaarden komen uit het wiskundig domein. In tegenstelling tot gewone relationele databases kunnen nu ook oneindige verzamelingen uitgedrukt worden.

Dit zal geïllustreerd worden met een eenvoudig voorbeeld van het relationeel database model en constraints van (on)gelijkheden:
Beschouw een binaire relatie $R(x,y)$ over een bepaald domein D. Een tupel (a,b) kan gezien worden als een conjunctie van constraints van gelijkheden: $(x = a) \wedge (y = b)$.
In fig. 12 wordt een opsomming gegeven van de eindige instantie van R. De instantie is hier een unie van drie formules, die elk op zich één tupel van de relatie zijn:
$(x = a1 \wedge y = b1) \vee (x = a2 \wedge y = b2) \vee (x = a3 \wedge y = b3)$.

Een relationele query kan op een gelijkaardige manier geïnterpreteerd worden. Beschouw bijvoorbeeld de relationele (FO) query $\exists x\ R(x,y)$. Toegepast op bovenstaande instantie kan het resultaat beschouwd worden als
$\exists x\ ((x = a1 \wedge y = b1) \vee (x = a2 \wedge y = b2) \vee (x = a3 \wedge y=b3))$.

R

X	Y
a1	b1
a2	b2
a3	b3

fig. 12 – Een eindige instantie van R

De query *evalueren* betekent dat we de formule gaan omzetten in een verzameling tupels. Dit wil zeggen, gebruik makend van de terminologie van de wiskundige logica, dat we deze formule gaan omzetten in een formule in disjunctieve normaalvorm die enkel de variabele *y* gaat gebruiken. We willen enkel de variabele *y* omdat dit de enige variabele is in de formule die *vrij* is. Variabele *x* is gebonden aan de kwantor $\exists$ (Definitie 1.1).
Het gewenste resultaat kan verkregen worden door gebruik te maken van *quantifier elimination techniques*:
$(y = b1 \vee y = b2 \vee y = b3)$. De output relatie van queries zijn dus met andere woorden formules zonder kwantoren.
Wat wel opgemerkt moet worden is dat de kwantoren hier niet gewoon reiken over de mogelijke waarden die zich in de database bevinden. De output relatie is typisch oneindig en de mogelijke waarden, die ingevuld kunnen worden door de *gebonden variabelen,* kunnen reiken over heel $\mathbb{R}$.

Uiteraard zijn we in het constraint model niet beperkt tot gelijkheden, ook ongelijkheden ($\neq$, <, >) mogen gebruikt worden.
Veronderstel dat het eerste tupel van ons voorbeeld nu volgende formule is: $(x \neq a1 \wedge y = b1)$. Door de ongelijkheid hier kan men hier te maken hebben met een oneindige relatie, in tegenstelling tot de instantie van fig. 12. De query $\exists x\ R(x,y)$ kan echter nog steeds geëvalueerd worden op relatie R op juist dezelfde manier [KLP98].

3.2 Het Constraint Databasemodel

We geven nu enkele voorbeelden van soorten constraints over de verzameling van de reële getallen.

Dense order constraints zijn van de vorm $x\ \theta\ y$ of $x\ \theta$ c, met x en y als variabelen, c $\in \mathbb{Q}$ als constante en θ als een van de operatoren $<$, $>$, $\leq$, $\geq$, $=$ of $\neq$.
Lineaire gelijkheid constraints zijn van de vorm $a_1x_1 + \ldots + a_nx_n = c$ en *lineaire ongelijkheid constraints* zijn van de gelijkaardige vorm $a_1x_1 + \ldots + a_nx_n\ \theta\ c$. Hier zijn $a_1, \ldots, a_n$ en c constanten, θ is zoals hiervoor. Deze constraints kunnen oneindige verzamelingen uitdrukken. Bijvoorbeeld: de constraint $x + y = 0$ representeert een rechte op het vlak die bestaat uit de verzameling van punten $\{ (a,-a) \mid a \in \mathbb{R} \}$. De constraint $x - y \geq 0$ stelt het halfvlak boven de rechte $x = y$ voor.

Polynomiale constraints veralgemenen de lineaire constraints door polynomialen van hogere graden toe te laten. Zo zal bijv. de constraint $x^2 + y^2 + z^2 \leq 1$ een bol uitdrukken met straal 1 in de 3-dimensionale ruimte.

In het constraint database model (CDB) veralgemenen we de notie van een tupel in een relatie met k argumenten naar de notie van een *"constraint tupel",* waarbij we een conjunctie van constraints bedoelen, over een bepaald constraint domein en over een gegeven verzameling met k variabelen. De veralgemening van het begrip van een relatie instantie is een verzameling $\{ t_1, \ldots, t_m \}$ van constraint tupels. De bedoelde semantiek van zo een instantie is de verzameling van alle punten $(a_1, \ldots, a_k)$ die voldoen aan de formule $t_1 \vee \ldots \vee t_m$.
Zoals we juist gezien hebben, kunnen constraint relaties een natuurlijke geometrische interpretatie hebben binnen bepaalde constraint domeinen. Dit is uiteraard de hoofdreden waarom we constraint databases kunnen gebruiken om ruimtelijke gegevens te modelleren en te querieën.
We geven in de volgende figuren het traditionele voorbeeld waarmee geïllustreerd wordt hoe oneindige geometrische objecten voorgesteld kunnen worden met behulp van verschillende klassen constraints.

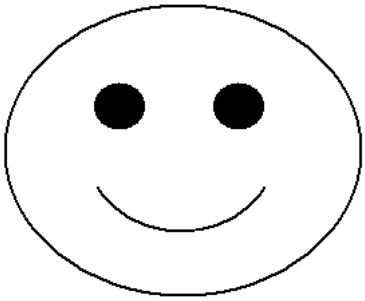


fig. 13 - Een voorbeeld van twee-variabelen polynomiale constraints

Het gezicht (op fig. 13) kan beschreven worden door polynomiale ongelijkheden, als de disjunctie van volgende tupels:
$(x^2/25 + y^2/16 = 1) \vee (x^2 + 4x + y^2 - 2y \leq 4)$
$\vee (x^2 - 4x + y^2 - 2y \leq -4) \vee (x^2 + y^2 - 2y = 8 \wedge y < -1)$.

De eerste gelijkheid beschrijft de buitenste ellips van de figuur, de tweede en derde ongelijkheid beschrijven de "ogen" en de laatste conjunctie van de disjunctie beschrijft de "mond."

Indien we niet met polynomiale ongelijkheden zouden mogen werken, maar enkel met lineaire aritmetische constraints, dan zou fig. 13 niet meer mogelijk zijn om uit te drukken. We kunnen daarentegen wel een lineaire benadering van het "gezicht" geven zoals in fig. 14.

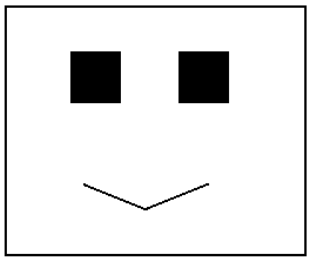


fig. 14 - Een voorbeeld van twee-variabelen lineaire constraints

Dit gebeurt dan met de volgende tupels:
$(-5 \leq x \leq 5 \wedge y = -4) \vee (-5 \leq x \leq 5 \wedge y = 4)$
$\vee (x = 5 \wedge -4 \leq y \leq 4) \vee (x = -5 \wedge -4 \leq y \leq 4)$
$\vee (-3 \leq x \leq -1 \wedge 0 \leq y \leq 2) \vee (1 \leq x \leq 3 \wedge 0 \leq y \leq 2)$
$\vee (3y = -x - 6 \wedge -2 \leq y \leq -1) \vee (3y = x - 6 \wedge -2 \leq y \leq -1)$.

De eerste 4 termen van de disjunctie beschrijven de buitenste rechthoek. De volgende twee deelformules van de disjunctie

beschrijven de vierkante “ogen” en de laatste 2 beschrijven de twee lijnstukken die samen de “mond” vormen.
In beide gevallen moet de volledige verzameling van punten die voldoen aan de constraints “opgeslagen” worden in de database. Het is uiteraard onmogelijk om al deze punten expliciet te stockeren, aangezien de verzamelingen van punten oneindig groot zijn. Het systeem slaat echter een eindige representatie op in de vorm van een verzameling constraints die de data beschrijven.
Het sleutelpunt is hier dat het gebruik van constraint volledig transparant is naar de gebruiker, die de data kan gebruiken *alsof* alle punten expliciet werden opgeslagen in de database.

3.3 Natural-active collapse

Constraint databases handelen niet meer enkel over eindige relaties zoals het traditionele relationele datamodel. Er kunnen nu ook oneindige relaties in de database voorkomen. Dit kan echter problemen veroorzaken voor het gebruik van kwantoren. Kwantoren, die reiken over oneindige ruimtes, zullen vanzelfsprekend moeilijker te evalueren zijn dan kwantoren die een eindig bereik hebben.
Voor dit probleem is er de *natural-active collapse* stelling. Deze stelling zal niet voor alle ruimtes gelden, maar wel voor de ruimte die ons interesseert: nl. ($\leq$, +, $\times$, 0, 1, $\mathbb{R}$). Dezelfde structuur zonder de optelling + heeft deze eigenschap dan weer niet.

Laten we een eindige database over een theorie U beschouwen. We kennen dan het *actief domein* van deze database, nl. de eindige verzameling van alle atomaire elementen van U die voorkomen in de relaties van de database.
Beschouw ook volgende query: $(\exists r)\ (\forall x,y)\ (R(x,y) \rightarrow (x\text{-}1)^2 + (y\text{-}1)^2 = r^2)$.
Laten we in dit geval de kwantoren reiken over heel $\mathbb{R}$.
De query drukt nu uit dat alle paren van punten in relatie *R* liggen op een bepaalde cirkel met de oorsprong als middelpunt. Het is cruciaal dat de existentiële kwantor in de query reikt over heel de verzameling van de reële getallen en niet enkel over het actieve domein van de database. Het is niet omdat alle paren op die bepaalde cirkel liggen dat de straal *r* van die cirkel zich ook in het actieve domein bevindt.
Het is nu echter wel mogelijk binnen FO[$\mathbb{R}$] om queries om te zetten zodat de kwantoren enkel reiken over het actieve domein. In het geval

van ons voorbeeld kan er bijvoorbeeld zo een equivalente query geschreven worden:
$(\exists x_0,y_0)(\forall x,y)\ (R(x,y) \rightarrow (\ (x\text{-}1)^2 + (y\text{-}1)^2\ = (x_0\text{-}1)^2 + (y_0\text{-}1)^2\ )$.
Het is perfect in orde dat de kwantoren enkel reiken over het actieve domein. Indien de kwantoren bij de eerste query ook enkel over het actieve domein zouden reiken, dan zou deze query *false* teruggeven. Terwijl de tweede query wel degelijk *true* aangeeft. Dit geeft dus een verschillende output terwijl beide queries eigenlijk equivalent zijn. Dit verstaat men onder de *natural active collapse:*
Voor elke formule ϕ bestaat er een formule ψ zodat voor elke eindige instantie *I(R)* geldt: $\phi(I(R)) = \psi^{active}(I(R))$ [VdB99].
Het kan bewezen worden dat in $(\leq,+,\times,0,1,\mathbb{R})$ alle queries zo omgezet kunnen worden [BL00].

We merken nog op dat er dus een onderscheid moet gemaakt worden met het uitvoeren van queries waarbij de kwantoren reiken over het actieve domein en de constraint tupels in databases waarbij variabelen kunnen reiken over heel $\mathbb{R}$.

3.4 Principe van implementatie

Om queries te schrijven voor constraint databases gebruiken we nu FO. Het idee hierbij is eenvoudig: queries worden geëvalueerd door instanties van database predikaten te vervangen door relaties, vervolgens kwantoreliminatie toe te passen en het resultaat om te zetten naar disjunctieve normaalvorm.

We herinneren ons de belangrijke eigenschappen uit §1.1 van de standard relationele querytalen (closure, efficiënte evaluatie). Het doel is om querytalen voor constraint databases te bepalen die deze eigenschappen ook hebben. Het basisidee zal zijn dat een query over een constraint database een eerste orde formule is over het constraint domein. Aangezien constraint relaties een (on)eindige verzameling punten voorstelt, zal het resultaat van een query de toepassing van de formule op die verzameling punten zijn.
In praktijk zal dit soort resultaten niet aanvaardbaar zijn, aangezien het doel van de introductie van constraints verloren zal gaan: nl. grote verzamelingen data voorstellen op een compacte manier. Voor dezelfde reden als waarom relationele query talen *gesloten* moeten zijn, willen

we hier dus ook dat het resultaat van een constraint query behoort tot dezelfde klasse van constraints als de input [KLP98].

Voorbeeld. We zullen het principe laten zien met behulp van twee snijdende rechthoeken zoals in fig. 15. De database in dit voorbeeld zal een ternaire relatie *R(z,x,y)* bevatten waarmee aangeduid wordt dat *(x,y)* een inwendig punt is <u>in</u> een rechthoek met naam *z*. Indien we willen zoeken naar alle paren van rechthoeken die elkaar snijden, dan kan dit uitgedrukt worden met behulp van volgende query:
$Q(z,z') \equiv z \neq z' \wedge (\exists x,y)\ (R(z,x,y) \wedge R(z',x,y))$.

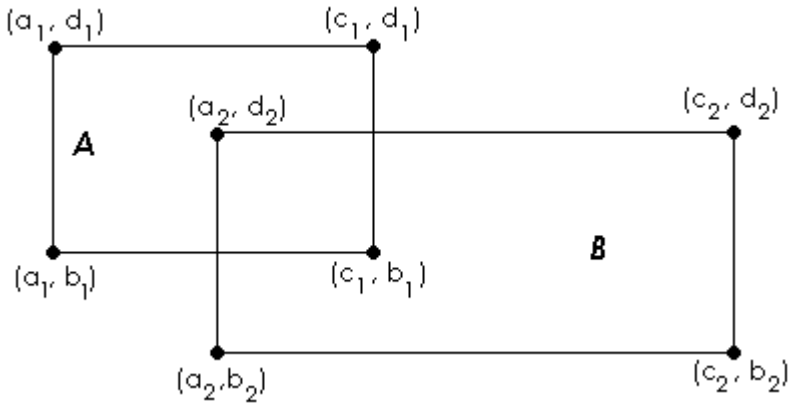


fig. 15 - Voorbeeld van 2 snijdende rechthoeken

De rechthoeken *A* en *B* worden in onze database beschreven door de constraint tupels $(z = n) \wedge (a \leq x \leq c) \wedge (b \leq y \leq d)$ waarbij we a,b,c en d moeten vervangen door de coördinaten van *A* resp. *B*. Dit wordt dus a_1 resp. a_2, b_1 resp. b_2,…We gebruiken hier dus enkel rechthoeken evenwijdig met de assen.

De query Q wordt dan geëvalueerd door elk voorkomen van *R* in de query te vervangen door de "waarde" van *R*. Met de "waarde" van een relatie bedoelen we de disjunctie van al zijn constraint tupels.
Het resultaat van onze query Q is hier dan alle paren van namen *(z,z')* die voldoen aan de formule:

$$z \neq z' \ \wedge\ (\exists x,y)\left(\bigvee_{(n,a_n,b_n,c_n,d_n)} (z = n \ \wedge\ a_n \leq x \leq b_n \ \wedge\ c_n \leq y \leq d_n\right)$$

$$\wedge\ \left(\bigvee_{(m,a_m,b_m,c_m,d_m)} (z = m \ \wedge\ a_m \leq x \leq b_m \ \wedge\ c_m \leq y \leq d_m)\right)$$

Deze grote disjuncties lopen hier dus over alle tupels van de relatie. Het resultaat van Q is daardoor een eindige relatie, in dit geval een verzameling van paren van rechthoeknamen.

We beschouwen nu volgende query Q':
$Q'(x,y) \equiv (\exists z,z')\ (z \neq z' \wedge R(z,x,y) \wedge R(z',x,y))$.

Deze query beschrijft de verzameling van alle punten *(x,y)* die in tenminste in twee disjuncte (d.i. volledig gescheiden) rechthoeken vallen. We merken op dat deze relatie zichzelf kan beschrijven door constraint tupels over hetzelfde constraint domein, zoals wordt geëist door de closure voorwaarde.

We zullen nu kijken hoe de resultaten voor Q en Q' nu effectief berekend worden. In het relationele model zou de query vertaald worden naar relationele algebra en dan (na optimalisaties) rechtstreeks berekend. Hier gaat dit echter geen oplossing bieden, want hierdoor ontstaat de vraag hoe algebraïsche expressies uitgevoerd zouden worden over een oneindige constraint relatie.
In principe zou dit echter wel kunnen met behulp van technieken uit de logica.

We merken de analogie op van een formule in disjunctieve normaal-vorm (DNF) waarbij elke disjunct een constraint tupel voorstelt. Zolang er een effectieve procedure te vinden is om een formule van een query om te zetten in een formule in DNF, hebben we een effectieve query evaluatie procedure.
Van de structuur $(\leq, +, \times, 0, 1, \mathbb{R})$ is geweten dat elke kwantorvrije formule omgezet kan worden in DNF [KUI02]. Dus om aan onze procedure te komen moeten we in staat zijn om kwantoren te elimineren uit een formule. Kwantoreliminatie is een belangrijk aspect in de logica. Er zijn verschillende theorieën bekend waarin kwantoreliminatie mogelijk is: dus waar elke formule omgezet kan worden in een equivalente formule zonder kwantoren. Zo weet men bijvoorbeeld dat dit mogelijk is bij de theorie $(\leq, +, \times, 0, 1, \mathbb{R})$.
Een bekend voorbeeld van kwantoreliminatie in deze structuur is de omzetting van volgend type formule:
$(\exists x)\ (ax^2 + bx + c = 0)$, de kwantorvrije formule is dan: $b^2 - 4ac \geq 0$.
Deze omzetting staat bekend als de berekening van de discriminant voor vierkantsvergelijkingen.

Het is ook belangrijk om te kijken naar de efficiëntie waarmee kwantoreliminatie gebeurt. In vele gevallen gebeurt dit met een hoge complexiteit. De complexiteit van kwantoreliminatie kan uitgedrukt worden in de kwantordiepte van een formule. Met de kwantordiepte bedoelen we de diepte die de kwantoren bereiken wanneer er een boom van de formule wordt opgesteld [KLP98].

We kennen nu al de eerste orde logica, we zijn vertrouwd met FOIES en constraint databases. In het volgende hoofdstuk zullen we het begrip topologische connectiviteit beschouwen om zo in het laatste hoofdstuk tot ons eigenlijke onderzoek te komen: kan topologische connectiviteit uitgedrukt worden met een FOIES?

Hoofdstuk 4

Topologische connectiviteit

In dit hoofdstuk wordt het begrip topologische connectiviteit toegelicht. Er zal een definitie gegeven worden en deze zal worden verduidelijkt met enkele voorbeelden. Vervolgens zal er bewezen worden dat topologische connectiviteit niet rechtstreeks in FO uitgedrukt kan worden.

4.1 Definitie en voorbeelden

Topologische connectiviteit kan gedefinieerd worden voor k dimensies (met $k \geq 2$). In ons onderzoek (in Hoofdstuk 5) zal de topologische connectiviteit onderzocht worden voor 2 dimensies.

Definitie 4.1 *k-dimensionele* topologische connectiviteit *(met $k \geq 2$) is een query over een relatie R met k argumenten. Het resultaat van de query zijn alle paren van punten in R die verbonden kunnen worden door een continue curve die volledig in R zit* [GKS98].

Definitie 4.1 wil met andere woorden zeggen dat topologische connectiviteit uitdrukt of twee punten met elkaar verbonden kunnen worden door enkel te bewegen binnen de figuren die in de database zitten. We merken ook op dat het begrip *continu* zoals we dit gebruikten in de definitie, niet overeenkomt met het concept *continuïteit* bij wiskundige functies. In onze context wordt hier mee bedoeld dat de figuren verbonden zijn, terwijl in de wiskunde 2 haakse op elkaar staande lijnstukken niet continu zullen zijn.

Er volgen nu enkele voorbeelden om een goed beeld te schetsen van topologische connectiviteit. We geven voorbeelden in het 2-dimensionale vlak omdat deze het eenvoudigst te tekenen zijn en dus ook het gemakkelijkst te begrijpen.

In elke figuur zullen de punten *x* en *y* topologisch geconnecteerd zijn met elkaar, terwijl punt *x* niet met punt *z* geconnecteerd zal zijn. Connectiviteit is een transitieve relatie, dus dit heeft als gevolg dat ook punt *y* niet topologisch geconnecteerd zal zijn met punt *z*.

Ons eerste voorbeeld (fig. 16) zal een database geven die willekeurige krommen bevat. Deze krommen kunnen elkaar snijden waardoor er onderlinge connectiviteit ontstaat. We merken op dat alle punten op een kromme uiteraard al onderling geconnecteerd zijn.

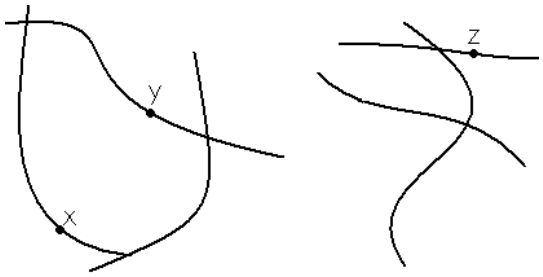


fig. 16 - Een database met krommen

De tweede database (fig. 17) die we zien bevat meetkundige vlakke figuren. Ook hier hebben we dezelfde situatie als daarnet: de figuren snijden elkaar waardoor er topologische connectiviteit ontstaat tussen de punten behorende tot de snijdende figuren. We moeten ook nog opmerken dat de punten van een figuur niet alleen kunnen slaan op de zijden maar eventueel ook op het gebied binnen of buiten de figuur.
De punten van de figuren op fig. 17 zijn enkel gedefinieerd op de zijden van de figuren. Bijgevolg is het puntenpaar (*x,z)* of het paar (*z,y*) ook hier niet topologisch geconnecteerd, want de cirkel waarop *z* ligt snijdt met geen enkele figuur van de database.

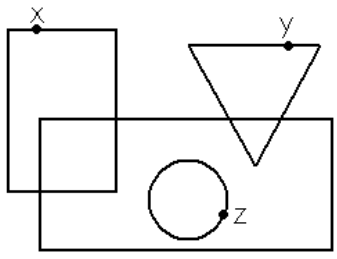


fig. 17 - Een database met meetkundige (niet-opgevulde) figuren

Als derde voorbeeld hebben we nu ongeveer voorgaande situatie genomen (zie fig. 18), maar nu is de rechthoek R volledig opgevuld bepaald. Dit wil zeggen dat alle punten die binnen de rechthoek liggen ook tot de database behoren. Het punt x op de cirkel is nu bijgevolg wel geconnecteerd met de rechthoek en dus ook met punt y op de snijdende driehoek. In het kleinere (afzonderlijke) rechthoekje zien we dat punt z uiteraard opnieuw niet geconnecteerd is met punt x of y. Het kleine rechthoekje is echter ook "opgevuld" bepaald en het binnenliggende punt w is bijgevolg wel topologisch geconnecteerd met punt z.

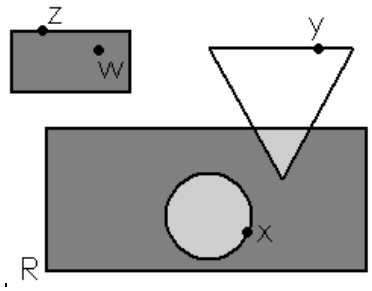


fig. 18 - Een database met meetkundige (en eventueel opgevulde) figuren

Als laatste geven we een voorbeeld van een database waarmee we zullen werken voor het onderzoek naar een FOIES. In dit onderzoek zal namelijk gewerkt worden met een database die volledig bestaat uit lijnstukken. De topologische connectiviteit is hier volledig analoog met voorgaande voorbeelden te bepalen. In fig. 19 zullen punten x en y opnieuw topologisch geconnecteerd zijn net zoals *(x,z)* en *(y,z)* puntenparen zijn die niet topologisch geconnecteerd zijn.

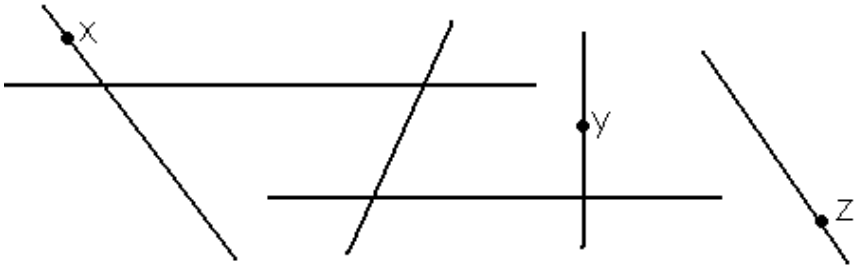


fig. 19 - Een database uitsluitend opgevuld met lijnstukken

4.2 Topologische connectiviteit is niet rechtstreeks uitdrukbaar in FO

Net zoals de transitieve sluiting van een graaf niet in FO uitdrukbaar is, kunnen we aantonen dat dit ook zo is voor topologische connectiviteit.
Om dit te bewijzen zal *topologische connectiviteit* gereduceerd worden naar een query waarvoor geweten is dat ze niet uitdrukbaar is in FO.

In deze paragraaf zal de reductie gegeven worden. De reductie zal gebeuren naar de *majority* query: hiervan werd bewezen dat deze query niet uitdrukbaar is in FO. Dit bewijs valt echter buiten deze masterproef en we zullen deze stelling dan ook als waar aannemen [GKS98].

Vooraleer er overgegaan kan worden tot de eigenlijke reductie zullen we *majority* query moeten definiëren.

Definitie 4.2 *De* majority query *krijgt twee eindige relaties R_1 en R_2 (met dezelfde argumenten) als input. De output geeft true als en slechts als $R_1 \subseteq R_2$ en $|R_2| \leq 2|R_1|$.*

In feite wordt hier gewoon uitgedrukt dat de majority query *waar* geeft wanneer de eerste verzameling volledig in de tweede verzameling zit en dat deze deelverzameling ook nog eens minstens de helft van alle elementen bestrijkt (m.a.w. dat de deelverzameling de *meerderheid* (=majority) van de elementen bestrijkt).

De majority query is niet rechtstreeks uitdrukbaar in FO. Indien we kunnen aantonen dat deze query reduceerbaar is naar topologische connectiviteit, kunnen we besluiten dat topologische connectiviteit ook niet rechtstreeks uitdrukbaar is in FO.

Stelling. De majority query is voor $k \geq 2$ FO reduceerbaar naar de k-dimensionele topologische connectiviteitsquery.

Bewijs. Het volstaat om enkel het geval $k = 2$ te beschouwen. Het bewijs is eenvoudig uit te breiden naar hogere dimensies.

We nemen R_1 en R_2 als input voor de majority query, waarbij: $R_1 = \{ a_1, a_2, \ldots, a_n\}$, $R_2 = \{ b_1, b_2, \ldots, b_m \}$ en $R_1 \subseteq R_2$ en $a_i, b_i \in \mathbb{R}$. De inclusie lijkt misschien een beperking, maar deze test kan zeer eenvoudig uitgedrukt worden. We mogen veronderstellen (zonder verlies aan algemeenheid) dat deze waarden geordend kunnen worden zodat:
$0 < a_1 < a_2 < \ldots < a_n$ en $0 < b_1 < b_2 < \ldots < b_m$.

Het reductie-algortime werkt intuïtief als volgt:
We plaatsen lijnsegmenten binnen een rechthoek (bepaald door de punten (0,0) en $(2b_m,a_n)$) zodat wanneer $2|R_1| \geq |R_2|$ (dus wanneer *majority true* geeft) geldt:

1. De lijnsegmenten zijn (topologisch) verbonden
2. De linkerbenedenhoek is (topologisch) verbonden met de rechterbovenhoek.

Hoe gebeurt deze constructie formeel:
We stellen $a_0 = b_0 = 0$.
De volgende lijnsegmenten worden vervolgens getekend:

- Het lijnstuk van (0,0) tot $(2b_m,0)$ → de lijn op X-as.
- Het lijnstuk van $(0,a_n)$ tot $(2b_m,a_n)$ → de bovenzijde van de rechthoek.
- Alle lijnstukken voor $1 \leq i \leq m$ en $1 \leq j \leq n$ van (b_{i-1},a_{j-1}) tot (b_i,a_j) en van $(b_m + b_{i-1}, a_{j-1})$ tot $(b_m + b_i, a_j)$. → de diagonale lijnstukken binnen de rechthoek.

Dit wordt duidelijk geïllustreerd in fig. 20 waarbij we R_1 met zes elementen hebben genomen en R_2 met vier elementen. Vier elementen vormen de meerderheid uit zes elementen, dus de majority query zal *waar* geven. De figuur duidt ook topologische connectiviteit aan tussen de twee relaties. In fig. 21 zien we hoe de constructie twee relaties niet topologisch connecteert, wanneer de deelverzameling niet de meerderheid uitmaakt van de andere relatie. R_1 heeft hier vijf elementen en R_2 twee elementen.

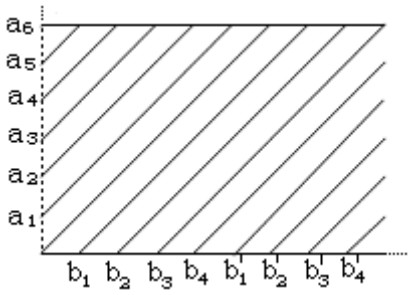


fig. 20 – Reductie waarbij de majority query *true* geeft

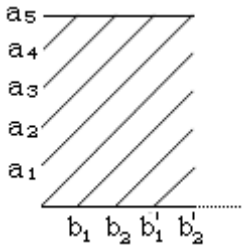


fig. 21 - Reductie waarbij de majority query *false* geeft

Deze reductie zorgt ervoor dat de topologische constructie volledig geconnecteerd is als en slechts als de majority query *true* zou geven op die relaties als input. Dus enkel wanneer er voldoende elementen zijn in de tweede relatie om het "plafond" van de eerste relatie te bereiken.

We merken op dat de geconstrueerde segmenten eerste orde definieerbaar zijn. We hebben dus een FO-omzetting ψ na ϕ:

$$R_1 \text{ en } R_2 \xrightarrow{\phi} \text{tekening} \xrightarrow{\psi} \text{ topologisch verbonden: true/false.}$$

Indien er dus een FO-query ψ zou bestaan die topologische connectiviteit uitdrukt, dan zouden we dankzij onze gevonden FO-query ϕ dus ook een FO-oplossing gevonden hebben voor de majority query. We weten echter dat deze query niet uitdrukbaar is in FO. Onze veronderstelling dat er een ψ kan bestaan, is dus met alle zekerheid fout. De topologische connectiviteitsquery kan niet in FO uitgedrukt worden [GKS98]. □

De figuren die in het bewijs zijn gebruikt, zijn heel algemeen bepaald. Het is zeker niet altijd het geval dat de lijnstukken zo mooi evenwijdig lopen. Dit heeft echter geen invloed op de constructie. Met behulp van een concreet voorbeeld kan dit nog beter verduidelijkt worden:

Voorbeeld. We bepalen de relaties concreet:
$R_1 = \{ 1, 3, 4, 7 \}$
$R_2 = \{ 1, 2, 3, 4, 7, 8 \}$

Met het blote oog kan al vastgesteld worden dat de majority query voor R_1 en R_2 *true* moet geven, want R_1 is een deel van R_2 en bevat vier elementen (wat de meerderheid is van de 7 elementen van R_2).
We zetten de gegevens nu uit volgens het reducerend algoritme op een tekening. De grafiek in fig. 22 geeft dan het resultaat van de reductie weer. Op de horizontale as staat (zoals het algoritme beschrijft) de punten van $2R_1$. Eerst worden de punten van R_1 gewoon uitgezet en daarna komen daar opnieuw alle punten van R_1 achter, maar nu telkens opgeteld met de laatste (en grootste) waarde van R_1.
R_1 is de *majority* van R_2 en de figuur moet topologisch verbonden zijn. Het is duidelijk op de figuur te zien dat dit ook inderdaad het geval is.

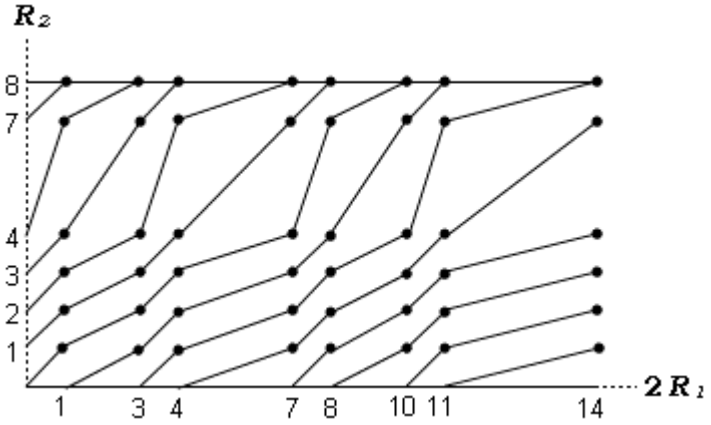


fig. 22 - Voorbeeld van een reductie

Hoofdstuk 5

Een FOIES voor topologische connectiviteit

Topologische connectiviteit is niet rechtstreeks uitdrukbaar in FO. Dat is het besluit dat we uit het vorige hoofdstuk kunnen trekken. Voor de transitieve sluiting van ongerichte grafen kwamen we eerder tot dezelfde vaststelling (§1.3), maar hiervoor werd tevens gevonden dat deze query incrementeel in FO kan onderhouden worden (§2.3). Waarom zou het dus niet mogelijk zijn om ook een FOIES te vinden voor topologische connectiviteit? Dit is de hoofdvraag waar deze masterproef zich mee bezig houdt.
In eerste instantie zullen we een FOIES proberen te vinden voor een eenvoudig ruimtelijk model. Dit model zal nl. geen willekeurige meetkundige figuren bevatten maar enkel lijnstukken. Later kan dit model dan nog worden uitgebreid. We kunnen al dadelijk de analogie opmerken tussen het probleem van de topologische connectiviteit van lijnstukken en de transitieve sluiting van grafen (fig. 23). De eindpunten van lijnstukken kunnen als knopen beschouwd worden en de lijnstukken zelf als bogen. Het grote verschil met ongerichte grafen is nu dat de snijpunten van de lijnstukken nieuwe knopen doen ontstaan.

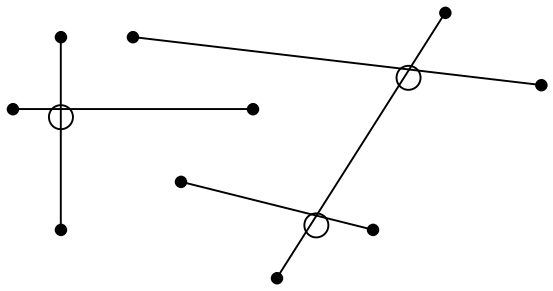


fig. 23 - Connectiviteit van lijnstukken

We kunnen de tekening dus ook beschouwen als een ongerichte graaf waarbij de snijpunten ook knopen zijn. De transitieve sluiting van deze graaf komt overeen met de connectiviteit van de lijnstukken.

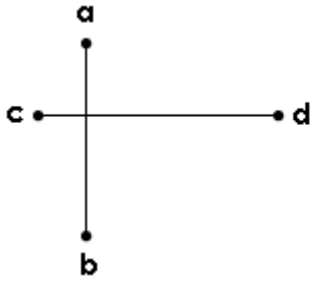
5.1 Database met lijnstukken

De ruimtelijke gegevens waarmee in het onderzoek zal gewerkt worden, bestaan enkel uit lijnstukken. We zullen nu enkele relaties afspreken waarmee consistent kan gewerkt worden en waarbij er geen verlies wordt geleden aan algemeenheid.
Verder is er ook noodzaak aan enkele hulprelaties, zoals het berekenen van lijnstukken die elkaar snijden. Ten slotte zal er ook nog gedefinieerd worden hoe we verwachten dat de query voor topologische connectiviteit uiteindelijk zal werken.

5.1.1. Relaties voor lijnstukken en punten

Lijnstukken kunnen gedefinieerd worden met constraints, net zoals andere meetkundige figuren dit kunnen. Wij zullen echter voor de eenvoudigheid de lijnstukken opslaan in onze database a.d.h.v. hun eindpunten. Bij het opstellen van de queries zal er wel gebruik gemaakt worden van de bijhorende vergelijkingen.

De ruimtelijke gegevens die in onze database zullen zitten zullen de unie zijn van allemaal rechte lijnsegmenten (bepaald door hun eindpunten).
Segment(p_1,p_2) is de relatie met alle lijnstukken [p_1p_2]. Een punt p_1 wordt bepaald door 2 coördinaten. Deze coördinaten bekomen we uit de relatie *Punt(p,x,y)* waarbij *p* de identifier is van het punt en de attributen *x* en *y* resp. de x- en y-coördinaat van dat punt.



Segment

Beginpunt	*Eindpunt*
a	b
c	d

Punt

Id	*x-coörd*	*y-coörd*
a	43	118
b	43	23
c	19	59
d	139	59

fig. 24 - Voorbeeld van een database met lijnstukken

5.1.2. Hulpqueries

Om regelmatig terugkerende gebeurtenissen te kunnen uitdrukken kan het zeer nuttig zijn om hulpqueries te definiëren. Zo een hulpquery kan dus eigenlijk gezien worden als een afkorting voor een bepaalde, veel terugkerende formule.

In het te voeren onderzoek voor de connectiviteit van lijnstukken zullen we vrij vaak moeten weten van een bepaald lijnstuk met welke lijnstukken uit de database het snijdt. Hiervoor zullen we een hulprelatie definiëren, net zoals we zullen doen om te kijken of een bepaald punt behoort tot een (en tot welk) lijnstuk van de database.

Vooraleer we overgaan tot het definiëren van de hulpqueries zullen we de vergelijkingen van de lijnstukken moeten bepalen aan de hand van hun eindpunten:
Veronderstel dat een lijnstuk gedefinieerd is door de eindpunten v en w met als coördinaten resp. v_x, v_y en w_x, w_y. Uit de meetkunde weten we dat de algemene vergelijking van een rechte gelijk is aan: $y = ax + b$ waarbij a de richtingscoëfficiënt (rico) is en b een constante [DJ97]. Evenzo weten we dat de rico berekend kan worden m.b.v. twee gegeven punten op de rechte (in casu onze eindpunten van het lijnstuk).

$$\text{rico} = a = \frac{w_y - v_y}{w_x - v_x}.$$

Dit geeft ons de voorlopige vergelijking:

$$\text{rechte} \leftrightarrow y = \frac{w_y - v_y}{w_x - v_x} x + b.$$

Constante b kan vervolgens berekend worden door één van de twee gekende punten in de vergelijking in te vullen en vervolgens b uit te rekenen:

$$b = v_y - \frac{w_y - v_y}{w_x - v_x} v_x.$$

Dit brengt ons tot een vergelijking met enkel bekende waarden voor de rechte met als vergelijking:

$$\text{rechte} \leftrightarrow y = \frac{w_y - v_y}{w_x - v_x} x + v_y - \frac{w_y - v_y}{w_x - v_x} v_x.$$

Deze vergelijking zou bij een oneindige rico problemen geven o.w.v. een deling door 0, daarom zullen de delingen uit de vergelijking weggewerkt worden, door de hele vergelijking met (w_x - v_x) te vermenigvuldigen:

$$\text{rechte } vw \leftrightarrow (w_x - v_x)y = (w_y - v_y)x + v_y + (v_y - w_y)v_x .$$

De lijnstukken waarmee gewerkt zal worden, zijn echter telkens maar een deel van de beschreven rechte in de vergelijking. Dit kan eenvoudig opgelost worden door een beperking op te leggen op de x en y-coördinaten. Deze beperking houdt in dat de punten van de rechten tussen de gegeven eindpunten moeten liggen.

Voorbeeld. Beschouw het lijnstuk bepaald door de punten (3,2) en (6,10). De constraint tupel die hier mee overeen zal komen is:
$3y = 8x - 22 \wedge 3 \le x \le 6 \wedge 2 \le y \le 10$.

De relatie, om te bepalen of een punt tot een lijnstuk in de database behoort, is nu eenvoudig te bepalen. Het punt moet in de vergelijkingen van de gekende lijnstukken ingevuld worden. Indien deze gelijkheid klopt en het punt valt binnen het opgegeven bereik van x- en y-coördinaten, dan zal dit punt behoren tot dat lijnstuk:

$$\begin{aligned} \textit{IsElemVan}(x,y,p,q) \quad \equiv \quad & \textit{Segment}(p,q) \wedge \\ & (\exists p_x, p_y, q_x, q_y)\ (\textit{Punt}(p,p_x,p_y) \wedge \textit{Punt}(q,q_x,q_y) \wedge \\ & (q_x - p_x)y = (q_y - p_y)x + p_y + (p_y - q_y)p_x \wedge \\ & p_x \le x \le q_x \wedge\ p_y \le y \le q_y . \end{aligned}$$

De argumenten voor deze relatie zijn eenvoudig te begrijpen: x en y zijn de punten waarvan we ons afvragen of zij behoren tot lijnstuk [pq]. We merken ook op dat de kwantoren van deze query reiken over het actieve domein en dat dit voldoende is voor de juiste uitvoer van de query.

De andere hulprelatie die gedefinieerd moest worden, is de query die bepaalt of twee lijnstukken elkaar snijden. Er worden dus vier punten gegeven als input. De query moet dan kijken of deze punten twee lijnstukken uit de database vormen en of deze lijnstukken ook effectief snijden. Uit de meetkunde weten we dat twee rechten elkaar altijd snijden tenzij ze evenwijdig zijn; met andere woorden er bestaat altijd een snijpunt tenzij ze gelijke rico's hebben en niet samenvallen. De eerste constraint die de query bijgevolg zal moeten opleggen is dat de

rico's van de rechten niet aan elkaar gelijk mogen zijn. Vervolgens kan gezocht worden naar het snijpunt *(x',y')* van de twee rechten. Aangezien de rico's niet gelijk zijn, is deze nl. steeds te vinden. We zullen nu een afleiding geven waardoor *x'* en *y'* kunnen bepaald worden met alleen vaste variabelen. Uiteindelijk moet nog gecontroleerd worden of het snijpunt van de rechten wel binnen het bereik van de lijnstukken ligt.

Afleiding snijpunt van 2 rechten (met ongelijke rico's). Twee rechten snijden elkaar in het punt dat behoort tot beide rechten; m.a.w. het punt dat de vergelijkingen van beide rechten aan elkaar gelijkstelt. Om dit punt te bepalen zullen we opnieuw vertrekken van de algemene vergelijking van rechten:
$R_1 \leftrightarrow y = a_1 x + b_1$
$R_2 \leftrightarrow y = a_2 x + b_2$
Beide rechten snijden elkaar in punt (*x',y'*), die bepaald kunnen worden op volgende manier:

$$\begin{gathered} a_1 x' + b_1 = a_2 x + b_2 \\ \Updownarrow \\ (a_1 - a_2)x' = b_2 - b_1 \\ \Updownarrow \\ x' = \frac{b_2 - b_1}{a_1 - a_2} \end{gathered}$$

Nu we de *x*-coördinaat van het snijpunt bepaald hebben is het bepalen van de *y*-coördinaat nog maar een kwestie van invullen:

$$y' = a_1 \frac{b_2 - b_1}{a_1 - a_2} + b_1$$

We hebben reeds berekend waaraan de parameters *a* en *b* gelijk zijn voor elk lijnstuk.

$$a = \frac{w_y - v_y}{w_x - v_x} \text{ en } b = v_y - \frac{w_y - v_y}{w_x - v_x} v_x$$

Voor elk lijnstuk zijn a en b vaste waarden, volledig bepaald door de eindpunten van de lijnstukken (zoals ook duidelijk in de formules van a en b te zien is). Snijpunt *(x',y')* kan men bijgevolg ook volledig bepalen aan de hand van de eindpunten van de lijnstukken. □

Er is ook nog een tweede geval dat we moeten bekijken; nl. wanneer een rico niet bestaat. Dit is het geval wanneer de rechte evenwijdig loopt met de Y-as.
Onze formule, die de relatie *Snijdt* zal definiëren, zal dus rekening moeten houden met deze gevallen.

In fig. 25 zien we de volledige opgestelde formule voor de relatie. We zullen de verschillende deelformules nu elk apart bespreken. Het eerste deel van de formule is eenvoudig te begrijpen. De opgegeven eindpunten moeten uiteraard lijnstukken definiëren die in de database zitten en de andere conjuncts dienen om de coördinaten op te vragen. De verschillende deelformules ϕ_i zullen elk een bepaald geval behandelen van hoe de lijnstukken liggen t.o.v. elkaar.

ϕ_1. Het eerste geval is wanneer beide lijnstukken gelijk lopen met de Y-as. Dus indien ($q_x = p_x \wedge v_x = w_x$) gelden dan weten we al dat beide rechten evenwijdig zijn met elkaar (en met de Y-as). Er zijn nu twee mogelijkheden. Beide rechten vallen samen en bestaat er een mogelijkheid dat ze minstens één gemeenschappelijk punt hebben (zie pq en vw op fig. 26). Ofwel vallen ze niet samen en kunnen we zeker zijn dat ze geen snijpunt hebben (pq en rs op fig. 26).

ϕ_2. Wanneer juist één lijnstuk evenwijdig (in dit geval *vw*) loopt met de Y-as (en de andere *pq* dus niet), dan kunnen we het snijpunt simpel bepalen door de vaste x-coördinaat p_x van *vw* in te vullen in de vergelijking van *vw* en zo zijn bijhorende y-coördinaat *y'* bepalen. Dit is het snijpunt. Rest ons nog om te checken of dit snijpunt binnen het bereik van lijnstuk *vw* ligt (fig. 27).

ϕ_3. Deze formule is volstrekt analoog met de vorige. Het behandelt het omgekeerde geval: *pq* is nu evenwijdig met de Y-as en *vw* heeft een andere richting.

$$
\begin{aligned}
\text{Snijdt}(p,q,v,w) \equiv\ & (\exists\ p_x,p_y,q_x,q_y,v_x,v_y,w_x,w_y)\ \text{Segment}(p,q) \\
& \wedge \mathit{Segment}(v,w) \wedge \mathit{Punt}(p,p_x,p_y) \wedge \mathit{Punt}(q,q_x,q_y) \\
& \wedge \mathit{Punt}(v,v_x,v_y) \wedge \mathit{Punt}(w,w_x,w_y) \wedge \\
\phi_1 \quad & [\ (q_x = p_x \wedge v_x = w_x) \\
& \qquad \rightarrow \quad (q_x = w_x \\
& \qquad\qquad \wedge (p_y \le v_y \le q_y \ \vee\ p_y \le w_y \le q_y)\ ] \\
\phi_2 \quad & \wedge [\ (q_x \neq p_x \wedge v_x = w_x) \\
& \qquad \rightarrow \quad ((\exists a_{pq}, b_{pq}, y') \\
& \qquad\qquad a_{pq} = \frac{q_y - p_y}{q_x - p_x} \wedge b_{pq} = p_y - \frac{q_y - p_y}{q_x - p_x} p_x \\
& \qquad\qquad \wedge\ y' = a_{pq} v_x + b_{pq}\ \wedge\ p_y \le y' \le q_y\ )\ ] \\
\phi_3 \quad & \wedge [\ (q_x = p_x \wedge v_x \neq w_x) \\
& \qquad \rightarrow \quad ((\exists a_{vw}, b_{vw}, y') \\
& \qquad\qquad a_{vw} = \frac{w_y - v_y}{w_x - v_x} \wedge b_{vw} = v_y - \frac{w_y - v_y}{w_x - v_x} v_x \\
& \qquad\qquad \wedge\ y' = a_{vw} p_x + b_{vw}\ \wedge\ v_y \le y' \le w_y\ )\ ] \\
\phi_4 \quad & \wedge [\ \neg(q_x = p_x \vee v_x = w_x) \\
& \qquad \rightarrow (\ (\exists a_{pq}, a_{vw}) \\
& \qquad\qquad a_{pq} = \frac{q_y - p_y}{q_x - p_x} \wedge a_{vw} = \frac{w_y - v_y}{w_x - v_x} \wedge a_{pq} = a_{vw} \wedge \\
& \qquad\qquad (\mathit{IsElementVan}(p,v,w) \\
& \qquad\qquad\quad \vee\ \mathit{IsElementVan}(q,v,w)\ )\ ] \\
\phi_5 \quad & \wedge [\ \neg(q_x = p_x \vee v_x = w_x) \\
& \qquad \rightarrow (\ (\exists a_{pq}, b_{pq}, a_{vw}, b_{vw}) \\
& \qquad\qquad a_{pq} = \frac{q_y - p_y}{q_x - p_x} \wedge a_{vw} = \frac{w_y - v_y}{w_x - v_x} \wedge a_{pq} \neq a_{vw} \wedge \\
& \qquad\qquad \wedge\ b_{pq} = p_y - \frac{q_y - p_y}{q_x - p_x} p_x \wedge b_{vw} = v_y - \frac{w_y - v_y}{w_x - v_x} v_x \\
& \qquad\qquad \wedge (\exists\ x', y')\ x' = \frac{b_{pq} - b_{vw}}{a_{vw} - a_{pq}} \\
& \qquad\qquad \wedge\ y' = a_{vw} \frac{b_{pq} - b_{vw}}{a_{vw} - a_{pq}} + b_{vw}\ \wedge p_x \le x' \le q_x \\
& \qquad\qquad \wedge\ p_y \le y' \le q_y \wedge v_x \le x' \le w_x \wedge v_y \le y' \le w_y\ )\ ].
\end{aligned}
$$

fig. 25 - De definitie van de relatie Snijdt

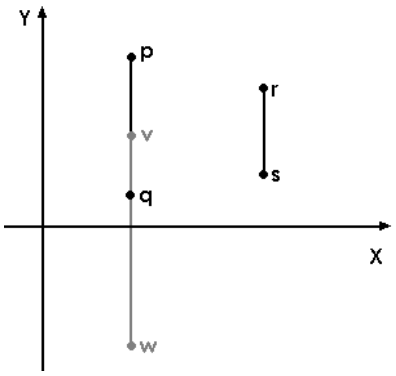


fig. 26 - Geval 1: beide lijnstukken evenwijdig met de Y-as

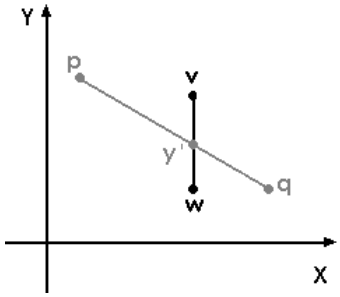


fig. 27 - Geval 2 (en 3): Juist één lijnstuk evenwijdig met de Y-as

ϕ_4. Het vierde geval is wanneer de rico's wel bestaan en de rechten samenvallen. In dit geval zijn de richtingscoëfficiënten uiteraard ook gelijk aan elkaar. We kunnen nu heel simpel een snijpunt vaststellen: er moet gewoon gecontroleerd worden of één van de eindpunten van één van beide lijnstukken een element is van het andere lijnstuk (fig. 28). De formule stelt eerst vast dat de rico's gelijk zijn en dan zal er een snijpunt vastgesteld worden als p of q een element is van $[uv]$.

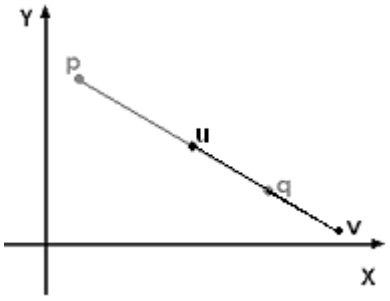


fig. 28 - Geval 4: Gedeeltelijk samenvallende lijnstukken (niet evenwijdig met de Y-as)

ϕ_5. De laatste formule behandelt alle andere gevallen; d.w.z. dat geen van beide lijnstukken evenwijdig loopt met de Y-as ($\neg(q_x = p_x \vee v_x = w_x)$) en dat beide lijnstukken niet evenwijdig zijn met elkaar. De volgende toekenningen die gebeuren zijn de a's en de b's van de vergelijkingen. De parameters a_{ij} zijn de respectievelijke rico's van de lijnstukken. In de volgende conjuncts worden de snijpunten van de rechten berekend:

$$x' = \frac{b_{pq} - b_{vw}}{a_{vw} - a_{pq}} \wedge y' = a_{vw}\frac{b_{pq} - b_{vw}}{a_{vw} - a_{pq}} + b_{vw}.$$

De laatste constraints dienen om te bepalen of het snijpunt *(x',y')* ook binnen het bereik van de lijnstukken valt. Zo kan het mogelijk zijn dat de rechten wel snijden, maar dat de lijnstukken niet tot in het snijpunt lopen. Op fig. 29 zien we gemakkelijk het triviale hier van in.

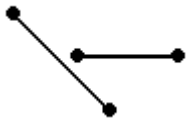


fig. 29 - Twee niet-snijdende lijnstukken

In de hele formule merken we enkele delingen op. Deze zijn gebruikt omdat het de leesbaarheid van de uitgebreide formule verhoogt. Toch moeten we opmerken dat onze structuur ($\leq,+,\times,0,1,\mathbb{R}$) geen delingen kent. Deze delingen worden ongedaan gemaakt door gewoon beide leden van de vergelijkingen te vermenigvuldigen met de voorkomende noemers. Hierdoor bekomen we uitdrukkingen zonder delingen. In deze deelformules kunnen nu variabelen voorkomen die reiken over heel $\mathbb{R}$ en hiervoor moeten we de natural active collapse stelling ter hulp roepen (§3.3). Deze zegt nl. dat elke formule van onze structuur kan omgezet worden naar een formule waarbij de variabelen reiken over het actief domein.

We zullen het principe van het elimineren van de delingen laten zien met deelformule ϕ_2:

$$a_{pq} = \frac{q_y - p_y}{q_x - p_x} \wedge b_{pq} = p_y - \frac{q_y - p_y}{q_x - p_x}p_x$$

$$\Downarrow$$

$$a_{pq}(q_x - p_x) = q_y - p_y \wedge b_{pq}(q_x - p_x) = p_y - (q_y - p_y)p_x$$

Uit deze resulterende formules kunnen we nu niet rechtstreeks de waarden van a_{pq} en b_{pq} afleiden, maar dit geeft niet. We weten namelijk dankzij de natural active collapse stelling dat deze formule omgevormd kan worden waarbij de gebonden variabelen weer reiken over het actief domein.

5.1.3. De topologische connectiviteitsquery

Het opzet is om na te gaan of twee willekeurige punten in het vlak elkaar kunnen bereiken door enkel te bewegen via lijnstukken in de database. Om dit te bereiken moeten de opgegeven punten al uiteraard zelf op een lijnstuk liggen. Vervolgens zal er gevraagd moeten worden of deze lijnstukken topologisch met elkaar verbonden zijn. Dit doet men door te vragen of één van de eindpunten van beide lijnstukken met elkaar verbonden zijn in de opgestelde connectiviteitsquery. Deze laatste query (hier de $\text{REACH}_{\text{connect}}$ query genaamd) is de eigenlijke query over enkele hulprelaties die zullen opgebouwd worden door het FOIES.

Indien we er in slagen om deze query incrementeel in FO up te daten, dan ziet onze topologische query er als volgt uit:

$$TopConnect(x_1,y_1,x_2,y_2) \equiv (\exists p,q,r,s) \quad IsElemVan(x_1,y_1,p,q) \wedge IsElemVan(x_2,y_2,r,s) \wedge REACH_{connect}(p,r).$$

5.2 Is connectiviteit reduceerbaar?

De analogie tussen het netwerk van lijnstukken en een ongerichte graaf is opvallend groot. Hierdoor ontstaat al snel het vermoeden dat topologische connectiviteit misschien te reduceren is tot het probleem van het onderhouden van de transitieve sluiting van ongerichte grafen. Na enkele pogingen hiertoe te hebben ondernomen heb ik jammer genoeg moeten besluiten dat dit niet mogelijk is.

Een mapping die gemaakt zou kunnen worden is door de eindpunten van de lijnstukken te beschouwen als knopen van de graaf. De

lijnstukken zelf vormen de bogen tussen de knopen. Indien er twee lijnstukken elkaar snijden, dan zal tussen de knopen overeenkomstig met de eindpunten van de lijnstukken ook een boog geconstrueerd worden (fig. 30).
Bij het toevoegen van een lijnstuk komt dit dus in eerste instantie neer op het toevoegen van die knopen en hun bijbehorende boog. Tot zover geen probleem voor de mapping. Nieuwe lijnstukken kunnen echter andere lijnstukken snijden, wat nieuwe, extra bogen met zich meebrengt. Bij het onderhouden van de transitieve sluiting wordt telkens maar een update gedaan voor het toevoegen van precies één boog. Een reductie naar dit probleem is bijgevolg niet mogelijk. Omdat er sprake is van mogelijk meerdere, nieuwe bogen in één toevoeging, kunnen beide problemen onmogelijk als equivalent beschouwd worden. Hierbij komt ook nog eens dat het nieuwe lijnstuk meerdere lijnstukken kan snijden, waardoor het aantal nieuwe bogen ook nog niet eens vastligt. Een reductie naar de transitieve sluiting van ongerichte grafen is dus jammer genoeg niet mogelijk. De analogie tussen beide structuren bestaat echter wel en zal later in het onderzoek ook nog uitgebuit kunnen worden!

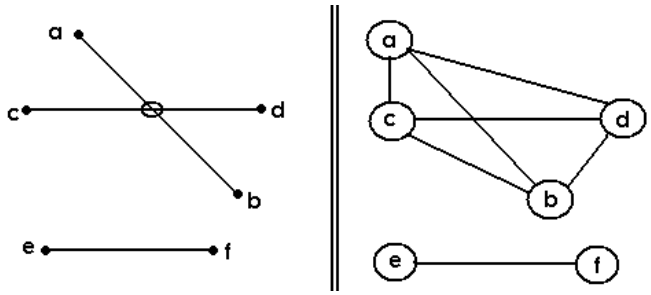


**fig. 30 - Links de lijnstukken in het vlak,
Rechts de gemapte ongerichte graaf**

5.3 FOIES voor topologische connectiviteit

Verschillende pogingen heb ik ondernomen alvorens te komen tot een FOIES voor topologische connectiviteit. In deze paragraaf zullen we de door mij gevonden constructie uitvoerig uit de doeken doen.

De gevonden FOIES is echter geen sluitende oplossing voor heel het probleem, in een volgende paragraaf zullen we de correctheid van de constructie aantonen, maar hierbij zullen enkele aannames moeten gemaakt worden. Deze aannames zullen de tekorten van onze constructie aantonen en ervoor zorgen dat we geen algemene oplossing voor het hele probleem gevonden hebben, maar wel een goede benadering.

5.3.1. Algemene aanpak

De analogie die we in §5.2 vastgesteld hebben tussen topologische connectiviteit en de transitieve sluiting van ongerichte grafen, gaat het basisidee vormen van de constructie van een FOIES voor topologische connectiviteit.
Vertrekkende van een lege ruimte zal er stap voor stap een wereld opgebouwd worden bestaande uit lijnstukken. In elke stap zal precies één lijnstuk worden toegevoegd of verwijderd. Steunende op het principe van Patnaik en Immerman zal er nu een ongerichte graaf worden opgebouwd die de topologische connectiviteit van de eindpunten van de lijnstukken weergeeft. Van deze graaf zal er incrementeel een spanning forest worden bijgehouden, zet zoals dat ook gebeurde bij Patnaik en Immerman.

De graaf zal opgebouwd worden zoals we dat besproken hebben bij onze poging tot reductie. Als er een lijnstuk wordt toegevoegd, dan zal er een boog bestaan tussen zijn eindpunten en tussen alle eindpunten van de lijnstukken die dit nieuwe lijnstuk snijden. Een boog tussen twee knopen in de graaf zal dus aanduiden dat de twee overeenstemmende eindpunten elkaar kunnen bereiken ofwel rechtstreeks via één lijnstuk ofwel via twee lijnstukken die elkaar snijden. Dit staat duidelijk in fig. 30 waar er een boog bestaat tussen a en b (omwille van lijnstuk [ab]) en tussen a en c (omdat [ab] gesneden wordt door [cd]). We zien dat twee snijdende lijnstukken een zogenaamde *clique* doen ontstaan tussen vier knopen.
Een toevoeging van een lijnstuk zal op deze manier voorgesteld kunnen worden in de gemapte graaf. Elk lijnstuk dat het nieuwe lijnstuk snijdt, zal telkens voor zo een *clique* zorgen in de graaf. We herinneren ons uit de graaftheorie dat een *clique* een verzameling knopen is in een ongerichte graaf, waarbij elk paar knopen in die verzameling verbonden is met een boog [Big74]. We zullen de spanning tree opbouwen in FO om deze componenten bij te houden. De spanning

tree zal er veel eenvoudiger uitzien omdat al die ingewikkelde *cliques* hier zullen verdwijnen.
Het voorbeeld uitgewerkt in fig. 31 laat zien hoe nauw geconnecteerde componenten in de gemapte graaf zullen samenhangen. Elk snijpunt zorgt voor 3 extra bogen in de graaf. Het alleenstaande lijnstuk [ef] vormt een samenhangcomponent op zich en heeft in de graaf maar één boog omdat er nog geen sprake is van snijpunten. Het veel eenvoudigere spanning forest wordt weergegeven in fig. 32.

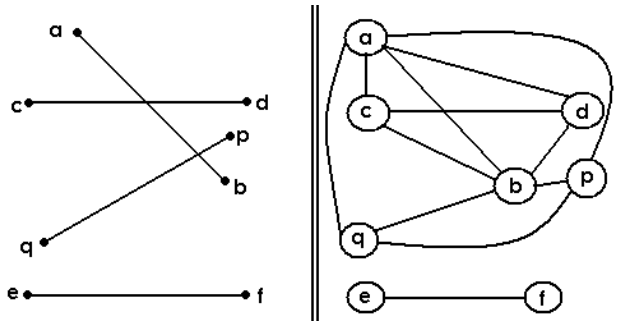


fig. 31 - Twee geconnecteerde componenten
links: De lijnstukken; rechts: de gemapte graaf

Voor het verwijderen van een lijnstuk zullen we analoog met Patnaik en Immerman te werk gaan. De link die in de spanning tree bestond en nu weg valt, zal proberen hersteld te worden via een andere bestaande boog om de twee samenhangcomponenten toch nog samen te houden (indien dit nodig is). We gaan hier dieper op in tijdens de constructie.

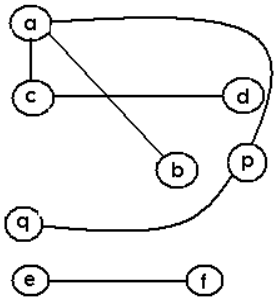


fig. 32 - Een mogelijke spanning forest van de database

5.3.2. Definitie van de gebruikte relaties in de constructie

Onze constructie zal gebaseerd zijn op de constructie die we gezien hebben bij Patnaik en Immerman (§2.3.2). Sommige relaties zullen

terugkeren (o.a. PV, F, P) maar met een iets andere invulling. De globale interpretatie zal wel gelijk blijven:

$\hat{F}(x,y)$. De relatie die de bogen van het spanning forest bevat. De interpretatie van deze relatie blijft dus ongewijzigd. Het spanning forest is een ongerichte graaf. Deze ongerichtheid telkens uitschrijven bezorgt ons veel overhead. Daarom gebruiken we volgende afkorting: $F(x,y) \equiv \hat{F}(x,y) \vee \hat{F}(y,x)$.

$\hat{PV}(x,y,z)$. Strikt genomen drukt deze relatie de paden uit die bestaan binnen het spanning forest. De relatie drukt uit dat knopen x en y elkaar kunnen bereiken in het spanning forest via de knoop z. De knopen hier zijn gemapte eindpunten van lijnstukken. Ook hier zal de eigenschap van ongerichtheid afgekort worden tot:

$PV(x,y,z) \equiv \hat{PV}(x,y,z) \vee \hat{PV}(y,x,z)$.

$P(x,y)$. De betekenis van deze relatie zal gedeeltelijk wijzigen. De punten x en y moeten hier niet noodzakelijk alle punten zijn die behoren tot de database. Wat wel vereist is dat de punten x en y behoren tot een bepaalde rechte die al in de database zit. Verder zal de relatie uitdrukken dat beide punten topologisch geconnecteerd zijn met elkaar. Deze query zal dus uiteindelijk de oplossing zijn voor het probleem van topologische connectiviteit.

$P(x,y) \equiv (\exists\, u_1,u_2,v_1,v_2)$ $\quad IsElemVan(x,u_1,v_1) \wedge IsElemVan(y,u_2,v_2) \wedge PV(u_1,u_2,u_1)$.

Verder zullen we ook enkele hulpqueries definiëren om later veel gemakkelijker het overzicht te behouden over onze gebruikte formules:

MinSnijlijn(x,y,a,b). Indien een lijnstuk ab gesneden wordt door meerdere lijnstukken die tot dezelfde samenhangcomponent behoren, dan geeft deze relatie het minimale lijnstuk hiervoor terug. Dit is belangrijk voor onze constructie omdat we bij meerdere mogelijkheden toch slechts één snijdend lijnstuk mogen gebruiken.

$MinSnijlijn(x,y,a,b) \equiv (\forall\, w,z)\ Snijdt(w,z,a,b) \wedge Segment(a,b) \wedge (w < x \ \vee (w = x \wedge z < y)) \rightarrow \neg P(x,w)$.

De formule is gemakkelijk te begrijpen. Lijnstuk [xy] is het minimale lijnstuk want voor alle andere lijnstukken [wz] die ook [ab] snijden geldt:

ALS hun eindpunten toch kleiner zouden zijn dan die van [*xy*] ($w < x \vee (w = x \wedge z < y)$), DAN wil dat zeggen dat beide lijnstukken niet tot dezelfde samenhangcomponent behoren. ($\neg P(x,w)$).

BoogInGraaf(x,y). In onze database zullen de segmenten (bepaald door hun eindpunten) worden bijgehouden evenals de spanning tree van de gemapte graaf. De gemapte graaf zelf zal niet expliciet worden bijgehouden in een relatie. De benodigde informatie kan gehaald worden uit de segment-gegevens. Om uit te drukken of er een boog bestaat van punt *x* naar punt *y* in de gemapte graaf zijn er twee mogelijkheden waaronder de boog zou kunnen bestaan:

1. Lijnstuk [*xy*] bevindt zich in de database en dus ook in de relatie *Segment*.
2. De boog *(x,y)* duidt er op dat eindpunt *x* het eindpunt *y* kan bereiken doordat de lijnstukken waartoe zij behoren elkaar snijden.

Deze relatie zal enkel nodig zijn wanneer er een lijnstuk verwijderd wordt. De formule is eenvoudig op te stellen:

$$BoogInGraaf(x,y) \equiv Segment(x,y) \vee [\,(\exists u,v)\ Segment(x,u) \wedge Segment(y,v) \wedge Snijdt(x,u,y,v)\,].$$

5.3.3. Toevoegen van lijnstuk [ab]

Voor elke update van de relaties zullen we eerst omschrijven en uitleggen wat er moet gebeuren. Later zullen we de formules geven en toepassen op de beschreven situaties. Deze aanpak zal het begrijpen van de constructie gemakkelijker maken.

Update van het spanning forest F(x,y).

De toevoeging van een lijnstuk [ab] kan impact hebben op onze bestaande spanning tree (herinner: we werken incrementeel, dus vertrekkende van een lege relatie, zullen we telkens over een geldige spanning forest beschikken bij elke stap).
Indien de eindpunten *a* en *b* beiden nog niet behoorden tot dezelfde spanning tree in het forest, dan zal er een boog gelegd moeten worden in de spanning tree van a naar b. Intuïtief zal gedacht worden dat dit steeds nodig zal zijn aangezien de kans klein is dat de lijnstukken telkens in dezelfde punten zouden vertrekken. Dit is inderdaad

meestal het geval, maar een tegenvoorbeeld is bijv. het construeren van een driehoek met 3 lijnstukken waarbij de derde toevoeging overbodig zou (fig. 33). In het geval dat a en b reeds in dezelfde component zitten, zal er voorlopig niets aan de spanning tree moeten gebeuren.

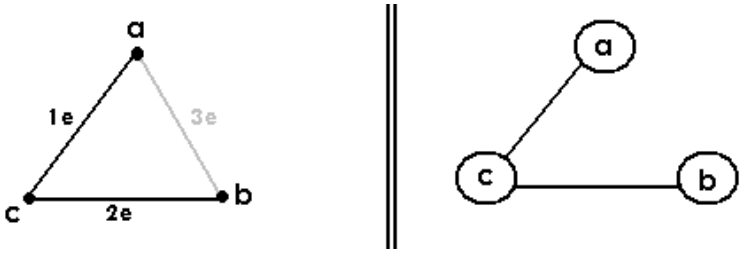


fig. 33 – Links. Driehoek abc met ab als laatste toegevoegd.
Rechts de spanning tree na het toevoegen van de 3 lijnstukken

Deze eerste stap verliep nog volstrekt analoog met de constructie van Patnaik en Immerman. Het verschil is nu echter dat een nieuw lijnstuk ook andere lijnstukken kan snijden en dat er op die manier nieuwe paden ontstaan. Voor elk lijnstuk zal er bepaald moeten worden of dit lijnstuk reeds tot een spanning tree van a of b behoorde. Indien dit niet het geval is, dan voegen we a toe aan de spanning tree van dat lijnstuk.

Indien één van beide punten (en het andere niet) tot de spanning tree van het lijnstuk behoorde, dan moet er niets gebeuren want de zojuist besproken stap zou dan uitgevoerd worden en voldoende zijn (in casu: het toevoegen van boog (a,b) aan het spanning forest).

Deze methode toepassen voor alle lijnstukken die met [ab] snijden zou ons in de problemen brengen. We herinneren ons dat een spanning tree acyclisch moet zijn. Indien we meerdere snijdende lijnstukken hebben die tot dezelfde samenhangcomponent behoren, dan zou de juist besproken handeling een cyclus kunnen creëren in de forestrelatie, dit moet dus vermeden worden. Het probleem kan vermeden worden door de handeling enkel toe te passen op het minimale lijnstuk van de snijdende lijnstukken van dezelfde samenhangcomponent.
We verduidelijken deze situatie met een voorbeeld in fig. 34. De lijnstukken [gh],[kl] en [cd] behoren tot één samenhangcomponent en de lijnstukken [ij] en [ef] tot een andere. Het nieuwe lijnstuk [ab] snijdt

de lijnstukken [kl] en [gh] van de ene component en [ij] van de andere component.

Noch a noch b behoorden tot een bestaande spanning tree, we hadden gezegd dat we nu a gingen toevoegen aan de spanning tree van dat lijnstuk. Door dit voor zowel [kl] als [gh] te doen (behorend tot dezelfde spanning tree), zou er een cyclus kunnen ontstaan. In de graaf op fig. 35 merken we op dat er een cyclus (a→l→k→c→g→h→a) ontstaat doordat we a koppelen aan beide lijnstukken. Indien we slechts met één lijnstuk zouden koppelen per samenhangcomponent zou er geen probleem zijn.

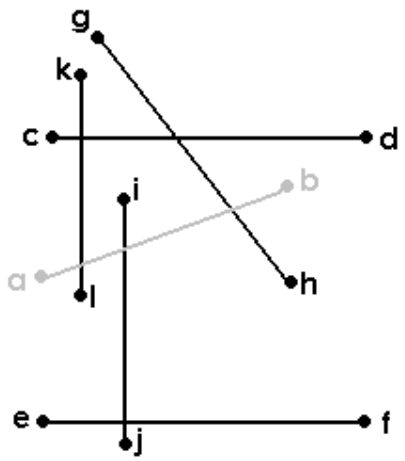


fig. 34 - Enkel de minimale snijlijn per samenhangcomponent behandelen

We merken daarom ook nog op dat in ons gegeven voorbeeld [kl] moet behandeld worden en ook [ij] want deze laatste zit in een andere samenhangcomponent.

De geüpdatet relatie zal er nu als volgt uitzien:

$$\begin{aligned}\hat{F}'(x,y) \equiv \quad & \hat{F}(x,y) \\ & \vee (\neg P(\mathrm{a},\mathrm{b}) \wedge ((x=\mathrm{a} \wedge y=\mathrm{b}) \vee (x=\mathrm{b} \wedge y=\mathrm{a}))) \\ & \vee (\quad (\exists u,v)\ Snijdt(\mathrm{a},\mathrm{b},u,v) \wedge \neg P(a,u) \wedge \neg P(b,u) \wedge u < v \\ & \qquad \wedge MinSnijlijn(\mathrm{a},\mathrm{b},u,v) \wedge ((x=\mathrm{a} \wedge y=u) \vee (x=u \wedge y=\mathrm{a}))).\end{aligned}$$

De formule is gemakkelijk op te splitsen in de verschillende gevallen die we juist behandeld hebben. Op de eerste regel wordt uiteraard de oude forestrelatie overgenomen. De tweede regel behandelt het geval waarbij a en b niet in dezelfde spanning tree zitten. Het overige deel van de formule drukt uit dat er nog bogen gelegd worden tussen a en

het kleinste eindpunt *u* van elk lijnstuk dat behoort tot een spanning tree waartoe noch a noch b behoren en dat lijnstuk is de minimale snijlijn met [ab] voor die samenhangcomponent.

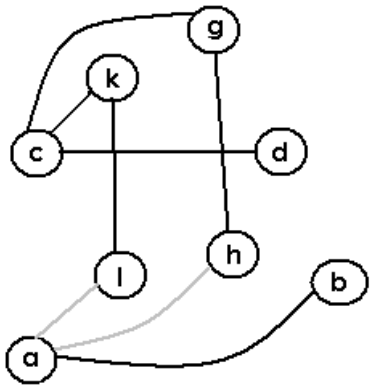


fig. 35 - Alle snijdende lijnstukken behandelen kan cycli veroorzaken in een spanning tree

Updaten van de relatie PV: de paden in het spanning forest.

Het spanning forest werd zojuist geüpdatet. De relatie *PV* die het verloop van de paden bijhoudt in het forest, moet nu uiteraard ook aan een update onderworpen worden.

De eigenlijke update kan ingedeeld worden in drie mogelijke gevallen. Elk geval zal in een aparte deelformule worden behandeld. De nieuwe relatie zal gedefinieerd worden door de formule:

$\hat{PV}'(x,y,z) \equiv \hat{PV}(x,y,z) \vee \phi_1 \vee \phi_2 \vee \phi_3$.

Voor het eerste geval beschouwen we de situatie waar a en b nog niet tot dezelfde component behoren ($\neg P(a,b)$). De boog (a,b) wordt dan toegevoegd aan het spanning forest. In eerste instantie komt hier dus het pad van a naar b bij via de knopen a en b:
$\phi_1 \equiv \neg P(a,b) \wedge [(x{=}a \wedge y{=}b \wedge (z{=}a \vee z{=}b)) \vee (x{=}b \wedge y{=}a \wedge (z{=}a \vee z{=}b))]$.

Verder werden er bogen geconstrueerd tussen a en de minimale snijlijn van elke *snijdende component*. De punten a en b krijgen nu een topologisch pad met elk punt van elke snijdende component via het eerste punt van elk minimaal lijnstuk. Met snijdende component bedoelen we elke component die minstens één lijnstuk bevat dat snijdt met [ab]. Dus in "symbolische" woorden omgezet: voor elke spanning

tree van een minimaal lijnstuk [uv], moet elk bestaand pad verlengd worden tot in a en b. Dit gebeurt door elke knoop *k* van die component (*PV(k,u,m)*) zijn pad tot *u* over te nemen en te verlengen tot in a en b. Dit doen we met de volgende formule:

$\phi_2 \equiv (\exists\ u,v,k,m)$ $\quad Snijdt(u,v,\text{a},b) \wedge MinSnijlijn(u,v,\text{a},\text{b}) \wedge PV(k,u,m) \wedge$
$[\ (x{=}k \wedge y{=}\text{a}) \vee (x{=}\text{a} \wedge y{=}k)\ ] \wedge (z{=}\text{a} \vee z{=}\text{b} \vee z{=}m).$

Het werk zit er nog niet op. Het linken van [ab] aan elke component is nu wel gebeurd, maar indien [ab] meer dan één lijnstuk snijdt, dan bestaat de mogelijkheid dat er ook verschillende samenhang-componenten nu topologisch verbonden zijn. In het spanning forest zijn deze samengebracht tot één spanningtree doordat vanuit a telkens een boog werd getrokken naar elke *snijdende component*. Dit is ook zeer duidelijk te zien in fig. 36 waar de grijze bogen de nieuwe bogen zijn in het spanning forest. De twee afzonderlijke componenten zijn nu verbonden met elkaar via knoop a. Knoop a kan hier als nieuwe root van de spanning tree beschouwd worden.

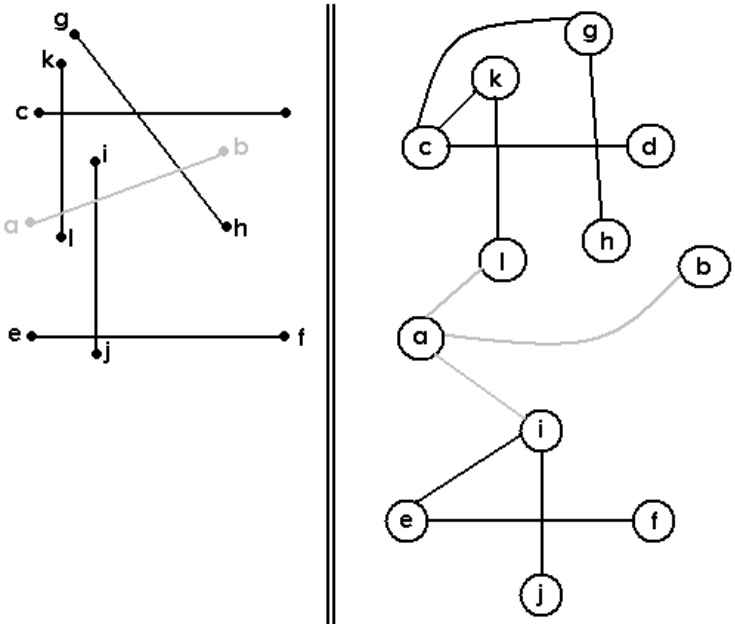


fig. 36 - Update van de spanning tree bij het toevoegen van [ab]

Opnieuw kijkend naar fig. 36 wil dit zeggen dat alle knopen van de bovenste component nu ook een pad moeten krijgen naar alle knopen van de onderste component gebruik makend van hun bestaande paden, met als extra link de knopen a en b. We merken hierbij op dat het zeer

belangrijk is dat ook knoop b wordt aangegeven als passerende knoop. Hier wijkt onze constructie dus lichtjes af van de zuivere constructie van Patnaik en Immerman. Het is echter noodzakelijk om dit te doen, want hiermee duiden we aan dat de link tussen beide componenten enkel kon gemaakt worden dankzij lijnstuk [ab]. Dit zal belangrijke informatie zijn voor later, indien we lijnstuk [ab] terug zouden willen verwijderen.

Hoe zal deze update nu in FO plaatsvinden: we selecteren alle paren minimale lijnstukken van elke *snijdende component* d.m.v. volgende formule:

$(\exists\ m,n,o,p)\quad Snijdt(m,n,\text{a,b}) \wedge Snijdt(o,p,\text{a,b}) \wedge MinSnijlijn(m,n,\text{a,b}) \wedge MinSnijlijn(o,p,\text{a,b}).$

De paden zelf worden vervolgens eenvoudig gelinkt. Elk combinatie van knopen heeft een pad dat loopt langs knoop *z*, waarvan we kunnen zeggen dat die *z* behoort tot de eerste component, tot de tweede component of het is een van de eindpunten a en b.
We krijgen voor deze veranderingen de formule ϕ_3:

$\phi_3 \equiv (\exists m,n,o,p,r)\quad Snijdt(m,n,\text{a,b}) \wedge Snijdt(o,p,\text{a,b}) \wedge MinSnijlijn(m,n,\text{a,b}) \wedge MinSnijlijn(o,p,\text{a,b}) \wedge (PV(x,m,r) \vee PV(y,o,r)) \wedge (z=\text{a} \vee z=\text{b} \vee z=r).$

5.3.4. Verwijderen van lijnstuk [ab]

Niet alleen toevoegen van lijnstukken maar ook het verwijderen moet ondersteund worden door onze FOIES. Hier zal opnieuw van het principe van Patnaik en Immerman worden vertrokken en worden aangepast aan onze constructie en mappende graaf.

Vooraleer we er formules bij gaan betrekken, zullen we aan de hand van ons voorbeeld beschrijven wat er precies zal moeten gebeuren bij het verwijderen van een lijnstuk.
Veronderstel dat we een situatie hebben zoals in fig. 37. Wanneer lijnstuk [ab] verwijderd wordt, dan zal de eerste stap van het update-proces overeenkomen met het verwijderen van boog (a,b) uit de spanning tree (indien deze in de spanning tree zat). Vervolgens zullen ook alle bogen naar de eindpunten van de lijnstukken, waarmee [ab] sneed, verwijderd worden (ook weer indien boog (a,b) in de spanning tree zat). Wanneer dit gebeurd is, kan men op zoek gaan naar een

lijnstuk dat de uiteengevallen spanning trees terug verbindt (voor zover deze zou bestaan). In ons voorbeeld zien we bijv. dat lijnstuk [ci] of [li] de twee componenten toch weer zou verbinden. Indien - zoals hier - meerdere lijnstukken in aanmerking komen om de verbinding te herstellen, dan kiest men zoals bij het toevoegen voor het *minimale* lijnstuk. De spanning tree wordt op deze manier hersteld zodat er een correcte mapping blijft bestaan tussen de graaf en de ruimte van lijnstukken.
Tenslotte moeten de gekende paden van de spanning tree ook worden geüpdatet.

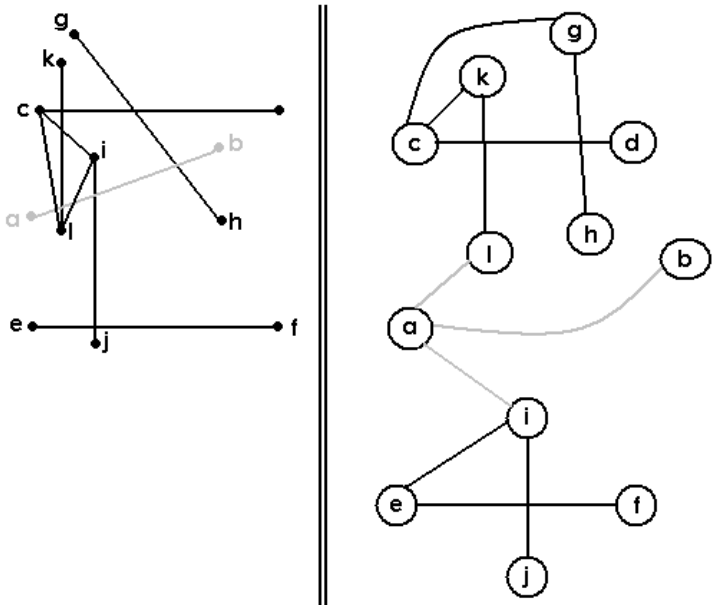


fig. 37 - Links: de ruimte met lijnstukken;
Rechts: de bijhorende spanning tree

Formule-gewijs zal de allereerste stap in het verwijderingproces het updaten van de relatie met de lijnstukken zelf zijn:
$Segment'(x,y) \equiv Segment(x,y) \wedge \neg ((x=a \wedge y=b) \vee (x=b \wedge y=b))$.

De tweede stap is het opstellen van de tijdelijke relatie T die de paden in de spanning tree zonder (a,b) voorstelt. Dit krijgt men door alle paden te verwijderen in het forest die zowel langs knoop a als knoop b passeren. Niet enkel de bestaande boog (a,b) moet verwijderd worden, maar ook de bogen die ontstaan zijn dankzij het lijnstuk [ab]. Dit zijn de bogen tussen a en de eindpunten van de snijdende lijnstukken met

[ab], indien boog (a,b) tenminste in de spanning tree zat. Dit heeft als effect dat we alle lijnstukken [*uv*] die snijden met [ab] moeten bekijken. Indien de overeenstemmende knoop *u* verbonden was met a (*Snijdt(u,v,a,b)* $\wedge$ *F(a,u)*), dan wil dat zeggen dat deze boog tot stand was gekomen dankzij lijnstuk [ab]. Alle paden die gebruik maken van beide knopen moeten dus eveneens verwijderd worden ($\neg$ (*PV(x,y,*a*)* $\wedge$ *PV(x,y,u)*). We herhalen dat deze maatregel enkel getroffen moet worden indien boog (a,b) in de spanning tree zat (Als F(a,b) geldt dan …).
Hierdoor komen we nu tot volgende formule voor relatie *T*:

$$T(x,y,z) \equiv PV(x,y,z) \wedge \neg(PV(x,y,a) \wedge PV(x,y,b)) \wedge F(a,b) \rightarrow (\exists\, u,v \quad Snijdt(u,v,a,b) \wedge F(a,u) \wedge \neg\,(PV(x,y,a) \wedge PV(x,y,u)\,).$$

De relatie *New,* die één of meerdere bogen zal zoeken om de uiteengevallen componenten weer samen te brengen, zal opgesteld moeten worden naargelang het aantal snijdende lijnstukken met [ab]. Deze beperking zal dan ook beletten dat onze constructie als algemene oplossing voor de topologische connectiviteit kan beschouwd worden.

Op het eerste zicht lijkt er zich geen probleem voor te doen. We zoeken gewoon naar een andere boog die de componenten toch weer verbindt. Dit kunnen we doen door te zoeken naar een boog in de gemapte graaf die twee uiteengevallen spanning trees terug met elkaar verbindt.
Dit zoeken gebeurt door voor twee snijdende lijnstukken van [ab] een boog te vinden in de gemapte graaf die de paden tussen de knopen (overeenkomstig met de eindpunten van die lijnstukken) terug met elkaar verbindt.
Dit principe lichten we nog even toe m.b.v. fig. 38. Het verwijderen van lijnstuk [ab] heeft als consequenties dat niet alleen boog (a,b) uit de spanning tree wordt verwijderd maar ook de bogen gelinkt aan de eindpunten van snijdende lijnstukken. In dit geval (a,u) en (a,w). We zien de spanning tree in twee delen uit elkaar vallen, maar we zien eveneens in de ruimte van de lijnstukken dat de figuur eigenlijk nog steeds topologisch verbonden is (dankzij [xy]). We gaan daarom op zoek naar een boog in de gemapte graaf (niet weergegeven op de figuur) die de twee delen terug met elkaar linkt.
De oplossing in ons voorbeeld zal één van de bogen (x,y), (z,y), (x,w) of (w,z) zijn. Deze bogen zitten in de gemapte graaf omdat zij het resultaat zijn van het snijden van lijnstukken [xz] en [yw]. Zoals steeds zullen we de boog selecteren door enkel de minimale eindpunten te

nemen. Veronderstel dat (x,y) de minimale boog is, dan zal deze boog de twee delen terug samenbrengen tot één spanning tree.

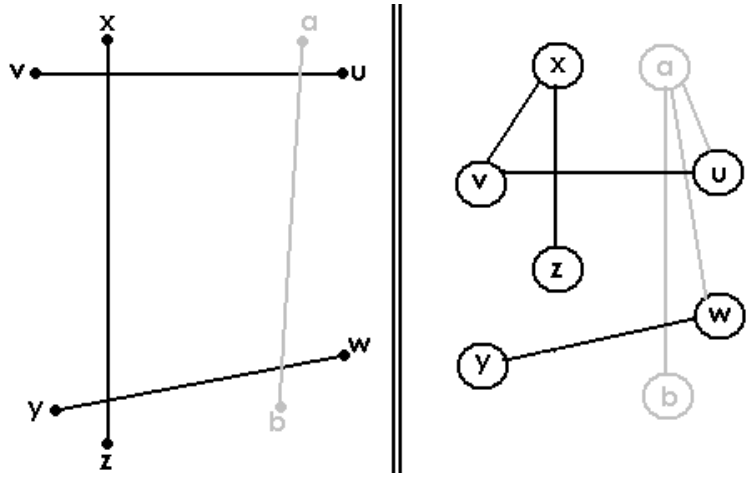


fig. 38 – Het verwijderen van [ab] doet de spanning tree in 2 delen breken Er moet hier nu een boog gevonden worden die de delen weer samen brengt

Waar ligt dan wel het probleem? Wel, de manier, zoals we ze nu beschreven hebben, houdt het risico in dat er een cyclus wordt gecreëerd in onze spanning tree. Bekijk de situatie in fig. 39. Rechts staat de spanning tree gegeven. De grijze bogen zijn de bogen verwijderd uit de spanning tree door het verwijderen van lijnstuk [ab]. De "volle" bogen vormen drie overblijvende delen van de spanning tree. Deze delen moeten terug gelinkt worden aangezien de figuur nog steeds topologisch geconnecteerd is. Met behulp van de zojuist uitgelegde methode zouden de "gestippelde" bogen gevonden en toegevoegd worden aan de spanning tree. We zien nu echter dat er een cyclus is ontstaan in de spanning tree (1-7-3-9-5-11-12-1), wat tegen de definitie indruist. Het zou de juiste werking van onze constructie vernietigen. Dit probleem wordt veroorzaakt doordat alle componenten *simultaan* teruggekoppeld worden. In ons voorbeeld was het voldoende geweest indien er slechts twee van de gevonden drie bogen werden toegevoegd aan de spanning tree. Men kan echter in FO niet kijken welke tupels al zijn toegevoegd in dezelfde stap om zo rekening te kunnen houden of een boog al dan niet mag worden toegevoegd. Hier vinden we de tekortkoming van onze constructie. We kunnen wel nagaan in dit specifieke geval dat er geen lus mag optreden. We kunnen dit omdat we weten dat er 3 componenten zijn en zo kunnen we checken of de verbindingen niet gelegd worden tussen reeds opnieuw verbonden componenten.

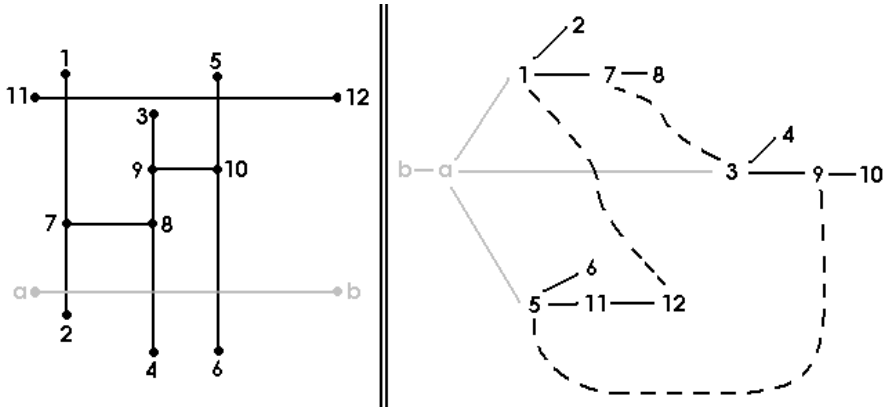


fig. 39 - De constructie kan een cyclus veroorzaken in de spanning tree

De methode zal aangepast moeten worden en er zal een deel van onze algemeenheid van de constructie opgegeven moeten worden. Hiermee bedoelen we dat onze constructie geen oplossing geeft voor het volledige probleem van topologische connectiviteit, maar dat er enkele assumpties moeten gemaakt worden bij het verwijderen (het toevoegen van lijnstukken zal wel in alle gevallen juist werken).

De aanpassing van de methode gebeurt in functie van het aantal snijdende lijnstukken (#*s*) met [ab]:

#s= 0 of 1. Indien er geen of één snijdend lijnstuk is met [ab], dan is er uiteraard geen probleem. Door slechts één boog toe te voegen, als koppeling tussen 2 verschillende spanning trees, is het onmogelijk om een cyclus te construeren.

$New_0(x,y) \equiv false.$

$New_1(x,y) \equiv (\exists u,v)\ Snijdt(u,v,\text{a,b}) \wedge \neg T(u,v,u) \wedge T(u,x,u) \wedge T(v,y,v) \wedge$
$BoogInGraaf(x,y) \wedge$
$(\forall p,q)\ BoogInGraaf(p,q) \wedge \neg T(p,q,p) \wedge T(u,p,u) \wedge T(v,q,v) \wedge$
$T(x,p,x) \wedge T(y,q,y) \wedge \neg T(p,q,p) \rightarrow x<p \vee (x=p \wedge y \leq p).$

#s=2. De oplossing voor twee snijdende lijnstukken is nog steeds juist. Het verwijderen van [ab] kan in dit geval de spanning tree in hooguit twee componenten doen breken. Indien er een boog gevonden wordt die beide componenten verbindt, dan zijn we nog steeds zeker dat de resulterende spanning tree acyclisch is aangezien één boog tussen 2 spanning trees, een nieuwe acyclische spanning tree doet ontstaan.

$New_2(x,y) \equiv (\exists u,v,w,z)\ Snijdt(u,v,\text{a,b}) \wedge Snijdt(w,z,\text{a,b}) \wedge$
$\neg T(u,w,u) \wedge T(u,x,u) \wedge T(w,y,w) \wedge$
$BoogInGraaf(x,y) \wedge$
$(\forall p,q)\ BoogInGraaf(p,q) \wedge \neg T(p,q,p) \wedge T(u,p,u) \wedge T(w,q,w)$
$\wedge T(x,p,x) \wedge T(y,q,y) \wedge \neg T(p,q,p) \rightarrow x<p \vee (x=p \wedge y \leq p).$

#s ≥ 3. Vanaf drie snijdende lijnstukken kunnen we niet meer met zekerheid garanderen dat er geen cyclus kan ontstaan. Indien de drie lijnstukken tot drie verschillende componenten behoren dan zou de situatie van fig. 39 bijvoorbeeld kunnen optreden. Daarom moet er aan de formule toegevoegd worden dat de toegevoegde bogen geen lussen in de componenten mogen creëren. De formule New_2 kan dus als basisformule gebruikt worden voor alle New formules van 2 of meer snijlijnen met [ab]. Voor elk aantal snijlijnen zal nu echter een bijkomende constraint moeten worden gedefinieerd die de acycliciteit van het spanning forest verzekert.
In het geval van 3: er zijn maximaal 3 componenten te vinden. Er zijn dus ook maximaal 3 koppels componenten te vinden. Er moet nu gezegd worden dat er slechts maximaal 2 van die bogen mogen gebruikt worden. We zijn zeker dat die twee bogen geen kwaad kunnen want dan zouden zij telkens dezelfde componenten linken. Onze formule is echter zo opgesteld dat enkel de kleinste van alle bogen, die dezelfde componenten linken, wordt genomen. Indien er een lus zou optreden, dan volgen we het omgekeerde principe: we sluiten de *maximale* (of de grootste) boog uit.
$BasisNew(x,y) \equiv New_2(x,y).$
$New_3(x,y) \equiv (\exists u,v,w,z)\quad BasisNew(u,v) \wedge BasisNew(w,z) \wedge$
$BasisNew(x,y) \wedge (x \neq u \vee y \neq v) \wedge$
$(x \neq w \vee y \neq z) \wedge (u \neq w \vee v \neq z) \wedge$
$[\ T(x,z,x) \wedge T(y,u,y) \wedge T(v,w,v)\ ]$
$\rightarrow [u > x \vee (u=x \wedge v > y) \vee$
$w > x \vee (w=x \wedge z > y)\ ]$

We lichten nog even de formule toe. We zeggen dus voor een boog *(x,y)* die in de basisrelatie zit: deze zit in de uiteindelijke relatie op voorwaarde dat, wanneer er zich een cyclus zou vormen door de bogen in de relatie dat deze boog dan niet de grootste boog uit die cyclusvormende boogverzameling is.
Hoe drukken we nu de voorwaarde van een eventuele cyclus uit? Dit gebeurt d.m.v. de deelformule [$T(x,z,x) \wedge T(y,u,y) \wedge T(v,w,v)$]. Indien

zowel de koppels knopen *[x,z], [y,u]* en *[v,w]* in dezelfde component zitten, dan weten we dat de bogen *(x,y), (y,u)* en *(v,w)* een cyclus zullen vormen (zie fig. 40 ter illustratie).

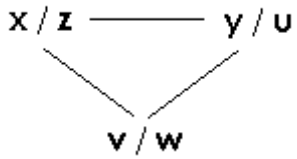


fig. 40 - Indien de knopen op deze manier tot componenten behoren, dan veroorzaken de bogen (x,y), (u,v) en (w,z) een cyclus

#s = 4. Bij vier snijdende lijnstukken met [ab] kan zijn spanning tree hooguit in vier componenten uit elkaar vallen. Dus de formule voor 3 snijdende lijnstukken moet nu uitgebreid worden naar de voorwaarde dat de eventuele 4 bogen geen cyclus mogen vormen. Het is echter ook mogelijk dat er geen vier maar drie bogen gevonden werden. Deze drie bogen zouden dan de vier componenten goed kunnen verbinden zonder een cyclus te vormen, maar het is ook mogelijk dat zij een cyclus vormen met drie componenten. Daarom moet ook nog de formule voor drie snijdende lijnstukken in rekening worden gebracht.

$$
\begin{aligned}
New_4(x,y) \equiv\ & New_3(x,y) \vee \\
& [(\exists u,v,w,z,s,t) BasisNew(u,v) \wedge BasisNew(w,z) \wedge \\
& \quad BasisNew(x,y) \wedge BasisNew(s,t) \wedge \\
& \quad (x \neq u \vee y \neq v) \wedge (x \neq s \vee x \neq t) \wedge (w \neq s \vee z \neq t) \wedge \\
& \quad (u \neq s \vee v \neq t) \wedge (x \neq w \vee y \neq z) \wedge (u \neq w \vee v \neq z) \wedge \\
& \quad [\, T(x,z,x) \wedge T(y,u,y) \wedge T(v,s,v) \wedge T(t,w,t) \,] \\
& \quad \rightarrow [u > x \vee (u = x \wedge v > y) \vee \\
& \qquad w > x \vee (w = x \wedge z > y) \vee \\
& \qquad s > x \vee (s = x \wedge t > y) \,].
\end{aligned}
$$

#s = n (met n > 2). In de twee voorgaande formules merken we duidelijk het iteratieve op in elke formule. We kunnen nu een formule definiëren voor *n* snijpunten (met n > 2). We merken op dat deze iteratieve formule geen formule is in FO. Het kan gezien worden als *n* verschillende FO-formules. Afhankelijk van hoeveel snijpunten er zijn, wordt een andere FO-formule gebruikt.

$$
\begin{aligned}
New_n(x,y) \equiv\ & New_3(x,y) \vee \ldots \vee New_{n-1}(x,y) \vee \\
& [(\exists u_1,v_1,\ldots,u_{n-1},v_{n-1})\ \ BasisNew(x,y) \wedge \\
& \qquad BasisNew(u_1,v_1) \wedge \ldots \wedge BasisNew(u_{n-1},v_{n-1}) \\
& \qquad \wedge (x \neq u_1 \vee y \neq v_1) \wedge \ldots \wedge (x \neq u_{n-1} \vee y \neq v_{n-1}) \\
& \qquad \wedge (u_1 \neq u_2 \vee v_1 \neq v_2) \wedge \ldots \wedge (u_1 \neq u_{n-1} \vee v_1 \neq v_{n-1}) \\
& \qquad \wedge (u_2 \neq u_3 \vee v_2 \neq v_3) \wedge \ldots \wedge (u_2 \neq u_{n-1} \vee v_2 \neq v_{n-1}) \\
& \qquad\qquad \vdots \\
& \qquad \wedge (u_{n-2} \neq u_{n-1} \vee v_{n-2} \neq v_{n-1}) \\
& \qquad \wedge [\ T(x,v_{n-1},x) \wedge T(y,u_1,y) \wedge \\
& \qquad T(v_1,u_2,v_1) \wedge T(v_2,u_3,v_3) \wedge \ldots \wedge T(v_{n-2},u_{n-1},v_{n-2})\ ] \\
& \qquad \rightarrow [\ u_1 > x \vee (u_1 = x \wedge v_1 > x) \vee \\
& \qquad\qquad \vdots \\
& \qquad\qquad u_{n-1} > x \vee (u_{n-1} = x \wedge v_{n-1} > x)\ ].
\end{aligned}
$$

Wanneer we de nieuwe bogen gevonden hebben voor het spanning forest, dan kunnen we deze al gaan updaten. De bogen van [ab] moeten verwijderd worden en de *new*-bogen moeten worden toegevoegd:

$$
\begin{aligned}
\hat{F}'(x,y) \equiv\ & [\ \hat{F}(x,y) \wedge \neg(x=\mathrm{a} \wedge y=\mathrm{b}) \wedge \neg(x=\mathrm{b} \wedge y=\mathrm{a}) \wedge \\
& (\exists u,v)\ Snijdt(u,v,\mathrm{a},\mathrm{b}) \wedge \neg(x=\mathrm{a} \wedge y=u) \wedge \neg(x=u \wedge y=\mathrm{a})\ ] \\
& \vee New_n(x,y) \vee New_n(y,x).
\end{aligned}
$$

Onze procedure eindigt nu met het updaten van de *PV*-relatie. De update vertrekt van de tijdelijke relatie *T (*relatie met alle paden van het forest zonder de bogen van [ab]*)*. De nieuwe paden die ontstaan door het linken van twee componenten met een boog uit de *New*-relatie.

$$
\begin{aligned}
\hat{PV}'(x,y,z) \equiv\ & T(x,y,z) \vee \\
& ((\exists u,v)\ New(u,v) \wedge T(x,u,x) \wedge T(y,v,y) \wedge \\
& \qquad (T(x,u,z) \vee T(u,v,z) \vee z=u \vee z=v \vee \\
& \qquad\qquad (\exists s,t)\ \neg Segment(u,v) \wedge Snijdt(s,u,t,v) \wedge \\
& \qquad\qquad\qquad (z=s \vee z=t\ ))).
\end{aligned}
$$

Het bepalen van de derde parameter vereist nog een beetje uitleg. Net zoals bij het invoegen zal zowel de *z* van de ene component, als de *z* van de andere component worden herbruikt. Verder moet ook de link *(u,v)* tussen de componenten worden aangegeven. De bogen uit de *new*-relatie kunnen nu echter ook een boog zijn die tot stand kwam door het snijden van twee lijnstukken. Dit zou betekenen dat het lijnstuk [uv] niet bestaat. Het is belangrijk voor onze constructie dat geweten is dankzij welk lijnstuk bepaalde connecties gemaakt zijn. Daarom zullen

we de andere eindpunten (*s* en *t* in de formule) van beide lijnstukken ook specificeren als *z*.

5.3.5. Evaluatie van de constructie

Het is eenvoudig na te gaan of alle gebruikte formules in FO zitten. Enkel bij het verwijderen hebben we een aanname gedaan waarbij handmatig (maar wel deterministisch) beslist moet worden met welke FO-formule de query wordt uitgevoerd.

Verder zullen we moeten nagaan of de onderhouden spanning tree, steeds een spanning tree zal blijven en aan zijn eigenschappen zal voldoen, waarbij de acyclische eigenschap de belangrijkste en meest cruciale is om te onderhouden.
Bij het toevoegen van een lijnstuk [ab] is dit eenvoudig na te gaan. Indien de eindpunten van het lijnstuk beiden nog niet tot een spanning tree in het forest behoorden, doet de constructie het volgende: men legt een boog tussen b en a en vanuit a construeert men een verbinding met elke spanning tree waarvan de component snijdt met [ab]. Indien er meerdere lijnstukken van dezelfde spanning tree [ab] snijden, dan wordt er slechts één lijnstuk gebruikt om de link te leggen (nl. de *"kleinste"*). Het is dus onmogelijk om een cyclus te krijgen aangezien a verbonden wordt met één of meerdere spanning trees, die onmogelijk een knoop kunnen delen want anders behoorden zij tot dezelfde spanning tree.
Indien a of b reeds tot een component behoorden, verandert er niet veel aan het principe, behalve dat er geen link meer gelegd wordt met die component waartoe ze reeds behoorde. Ook hier is de acycliciteit van het spanning forest gegarandeerd.
We kunnen hier al besluiten dat we een incrementeel FO-systeem hebben gevonden dat de topologische connectiviteit onderhoudt.

Helaas kunnen we niet hetzelfde zeggen voor het decrementele aspect van het probleem. Hier is slechts een beperkte oplossing gevonden, die enkel kan gebruikt worden m.b.v. een iteratieve programmeertaal. Er zal moeten opgevraagd worden welke rechten snijden met het te verwijderen lijnstuk [ab]. De teruggegeven tupels zullen geteld moeten worden en aan de hand hiervan zal er een query geselecteerd moeten worden om de update correct uit te kunnen voeren. Dit kan niet puur in FO gebeuren, hiervoor zal er een deterministische beslissing geno-

men worden om te bepalen welke FO-formule moet gebruikt worden. We merken dus op dat dit geen exacte oplossing is voor ons probleem en slechts een benadering, maar dat onze methode wel volledig geautomatiseerd kan worden. We zijn zeker dat de decrementele constructie werkt want de acyclische eigenschap van de spanning tree blijft steeds gegarandeerd. Alle mogelijke gevallen waarbij een cyclus zou kunnen optreden, zijn reeds behandeld tijdens het construeren van het systeem zelf (zie vorige paragraaf) en zal hier niet opnieuw behandeld worden.

Conclusie

Deze masterproef heeft zich toegelegd op een relatief jong onderzoeksgebied binnen de informatica, nl. de eerste orde in- en decrementele evaluatie. Dit zijn systemen die oplossingen voor problemen incrementeel (en/of decrementeel) bijhouden a.d.h.v. updates, meestal omdat de problemen te complex zijn om rechtstreeks de oplossing (in eerste orde logica) uit te drukken.

De masterproef spitste zich vooral toe op het probleem van het onderhouden van de topologische connectiviteit van ruimtelijke gegevens. Er werd gezocht naar een FOIES voor dit probleem. Hiervoor werd er eerst naar een eenvoudiger probleem (met enkele analogieën) gekeken: de transitieve sluiting van ongerichte grafen. Patnaïk en Immerman hadden hier een FOIES voor gevonden. Deze oplossing werd volledig uitgelegd en de onvolledigheden van de constructie van Patnaïk en Immerman werden opgevuld en vervolledigd.
De oplossing voor het probleem van topologische connectiviteit voor lijnstukken werd gebaseerd op de oplossing van Patnaïk en Immerman. Hiervoor heb ik namelijk een mapping kunnen construeren die de verzameling lijnstukken mapt op een ongerichte graaf. Zo was het mogelijk om opnieuw een spanning forest op te bouwen van die gemapte graaf. De constructie was veel complexer dan die van Patnaïk en Immerman want i.p.v. één boog kon een lijnstuk met meerdere andere lijnstukken snijden en voor meer topologische verbindingen zorgen dan die ene boog bij ongerichte grafen. Op deze manier heb ik voor het toevoegen van lijnstukken een systeem gevonden dat volledig in FO uitdrukbaar is. We kunnen hieruit besluiten dat topologische connectiviteit voor lijnstukken *first order incrementeel definieerbaar* is.
Voor het verwijderen van lijnstukken (decrementeel) heb ik vastgesteld dat de constructie van Patnaïk en Immerman te kort komt om een sluitende FOIES voor het verwijderen te construeren. Ik heb er dan voor kunnen zorgen dat dit systeem met een kleine deterministische automatisatie een FO-formule kan selecteren a.d.h.v. het aantal snijpunten van het te verwijderen lijnstuk, waardoor het systeem nog in FO werkt. Doch het systeem is door de automatisatie

geen volledig decrementeel FOIES. Doordat de constructie van Patnaïk en Immerman het gevaar op de cyclus in de spanning tree niet kan vermijden, kunnen we besluiten dat er geen volledig sluitend FOIES zal gevonden worden voor topologische connectiviteit m.b.v. Patnaïk en Immerman. Door deze vaststelling kan men ook vermoeden dat een gelijkaardig probleem van deze constructie ook terug zal opduiken bij andere constructies, waardoor het dus onmogelijk lijkt om een volledig sluitend FOIES voor topologische connectiviteit te vinden.

Als slot van deze masterproef wil ik er nog op wijzen dat mijn constructies voor het onderhouden van de topologische connectiviteit van lijnstukken enkel de ruimtelijke eigenschap van het snijden van lijnstukken gebruikt, waarbij er een volgorde ligt op de snijpunten.
De constructie zou dus geprojecteerd kunnen worden op complexere ruimtelijke gegevens dan lijnstukken, waarbij de relatie *Snijdt* anders zal gedefinieerd moeten worden. De oplossing zal hier liggen bij rationale krommen, dit zijn krommen die parametriseerbaar zijn. De snijpunten tussen zulke krommen kunnen hier aan de hand van een parameter t bepaald worden; de punten die gelijkvallen in de parametervergelijking op "tijdstip" t. Dit "tijdstip" zal voor alle betreffende krommen gelden en hiermee zal ook op een eenvoudige manier een volgorde op de snijpunten bepaald worden. We kunnen deze masterproef dus besluiten met de opmerking dat het mogelijk zou moeten zijn om de geconstrueerde FOIES uit deze thesis uit te breiden van lijnstukken naar rationale, parametriseerbare krommen.

Bibliografie

[Big74] Biggs N. (1974) “Algebraic Graph Theory”, Cambridge, Cambridge University Press.

[BL00] Benedikt M. en Libkin L. (2000) “Relational queries over interpreted structures”, *Journal of the ACM*. 47.

[DJ97] Daems J-P. en Jennekens E. (1997). *Analytische meetkunde*. Antwerpen, de boeck. 17-20.

[DS95] Dong G. en Su J. (1995) “Incremental and Decremental Evaluation of Transitive Closure by First-Order Queries”

[GKS98] Grumbach S, Kuper G. en Su J. (1998) “Expressive Power: The Infinite Case”, In: Kuper G, Libkin L en Paeredeans J. *Constraint Databases*. Berlin, Springer. 89-102.

[GS97] Grumbach S. en Su J. (1997) “Queries with arithmetical constraints”

[GUW02] Garcia-Molina H, Ullman J. en Widom J (2002). “The Relational Data Model”, In: *Database Systems: the complete book*. New Jersey, Prentice Hall. 61-68.

[GV05] Gyssens M. en Vansummeren S. (2005) “Fundamenten van Databases”, Diepenbeek, Universiteit Hasselt. 23-25.

[KLP98] Kuper G, Libkin L. en Paredaens J. (1998) “Introduction to Constraint Databases”, In : *Constraint Databases*. Berlin, Springer. 1-20.

[KUI02] Kuijpers B. (2002) “Logica en Modelleren”, Diepenbeek, Limburgs Universitair Centrum. 73-86.

[PI95] Patnaik S. en Immerman N. (1995) “Dyn-FO: A Parallell, Dynamic Complexity class”

[Ull82] Ullman J. (1982) “Relational Calculus”, In: *Principles of Database Systems (2nd edition)*. Maryland, Computer Science Press. 156-170.

[VdB98] Van den Bussche J. (1998) “Constraint Databases, Queries and Query Languages”, In: G. Kuper, L. Libkin en J. Paredaens, *Constraint Databases*. Berlin, Springer. 21-35.

[VdB99] Van den Bussche J. (1999). “Constraint Databases: A tutorial introduction”

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen en uw akkoord te verlenen.

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Het onderhouden van connectiviteit in ruimtelijke gegevens
Richting: **Master in de informatica** Jaar: **2006**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Deze toekenning van het auteursrecht aan de Universiteit Hasselt houdt in dat ik/wij als auteur de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij kan reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

U bevestigt dat de eindverhandeling uw origineel werk is, en dat u het recht heeft om de rechten te verlenen die in deze overeenkomst worden beschreven. U verklaart tevens dat de eindverhandeling, naar uw weten, het auteursrecht van anderen niet overtreedt.

U verklaart tevens dat u voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen hebt verkregen zodat u deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal u als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze licentie

Ik ga akkoord,

Stephen MARTENS

Datum: