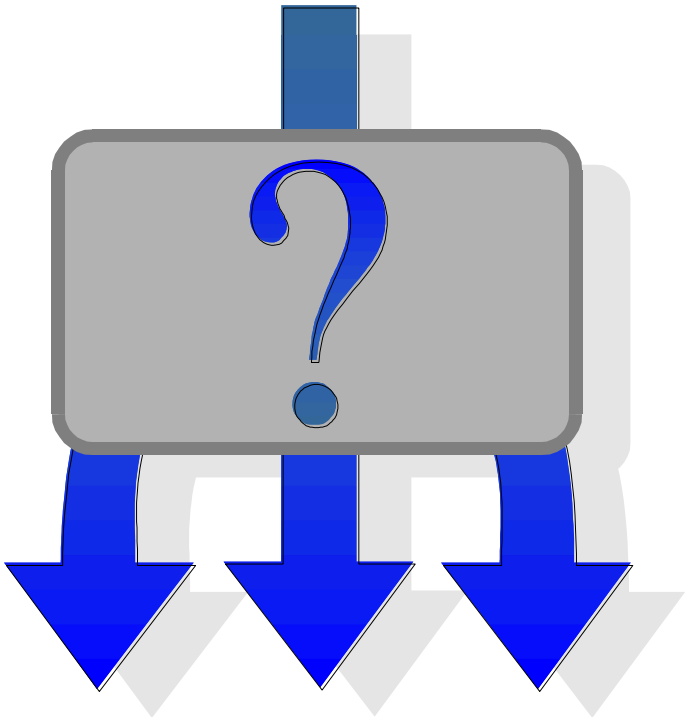
Parallelle Data Mining

met behulp van Dynamische Decompositie



Universiteit van Amsterdam *mei 1997*

Carel van den Berg Menzo Windhouwer

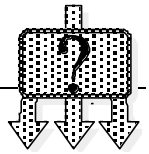
Parallelle Data Mining

met behulp van
Dynamische Decompositie

vrijdag 30 mei 1997

Student: M.A. Windhouwer
Collegekaartnummer: 9410066
Email: windhouw@wins.uva.nl

Begeleider: dr C.A. van den Berg



Voorwoord

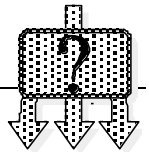
Het verslag dat u op dit moment in uw handen heeft bevat de weerslag van het onderzoek dat ik voor mijn afstuderen bij de studierichting Bedrijfsinformatiesystemen, specialisatie Technische informatiesystemen aan de Universiteit van Amsterdam faculteit Wiskunde, Informatica, Natuurkunde en Sterrenkunde heb uitgevoerd.

Zoals de meeste onderzoeken is ook dit onderzoek niet door één persoon uitgevoerd maar hebben veel mensen, direct of indirect, hun steentje eraan bijgedragen. In dit voorwoord wil ik daarom graag de mensen die mij bij dit onderzoek geholpen hebben bedanken.

Allereerst mijn begeleider Carel van den Berg, die altijd een positieve kijk behield op het verloop van het onderzoek en wiens vele opmerkingen een grote invloed op het eindresultaat hebben gehad. Daarnaast Peter Boncz voor het ontwikkelen en debuggen van zowel Monet als de gebruikte TCP/IP module. Zowel Carel van den Berg, Peter Boncz en Martin Kersten wil ik bedanken voor hun commentaar op de conceptversie van deze scriptie. Ook Marcel Worring en Dennis Koelma voor het commentaar bij de studentenbesprekingen (waarvan ik er onderhand een respectabel aantal heb mogen bijwonen), dat toch altijd weer hielp bij het bijsturen van het onderzoek.

Apeldoorn, vrijdag 30 mei 1997

Menzo Windhouwer



Samenvatting

Data mining houdt zich bezig met het ontdekken van nieuwe kennis in grote gegevensverzamelingen. Deze grote gegevensverzamelingen worden beheerd door database management systemen. Tijdens het data mining proces wordt deze verzameling veelvuldig geraadpleegd door vragen te stellen aan het database management systeem. Het beantwoorden van deze vragen door het database management systeem neemt veel tijd in beslag.

Om de responstijd van het beantwoorden van de vragen te versnellen kunnen meerdere processors tegelijkertijd aan het beantwoorden van een vraag werken. Dit kan door de gegevensverzameling over meerdere database management systemen te verdelen. Om nog een antwoord op de originele vraag te kunnen verkrijgen dient de originele vraag nu te worden verdeeld over de verschillende database management systemen. Het onderzoek richt zich op dit probleem, de onderzoeksvraag luidt dan ook als volgt:

> *Hoe verdeel ik een data mining vraag zo optimaal mogelijk over verschillende database servers, gegeven een bepaalde fragmentatie van de originele database, en voer ik deze deelvragen uit zodat er ook daadwerkelijk een reductie van de responstijd wordt verkregen?*

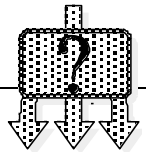
Het verdelen van een data mining vraag in deelvragen voor verschillende processors blijkt uit verschillende stappen te bestaan.

De eerste stap is het herschrijven van de originele vraag. Bij dit herschrijven vervangen we verwijzingen naar de originele database tabellen door de reconstructie regels om deze tabellen te herstellen uit de verschillende fragmenten. Deze reconstructie regels introduceren nieuwe operaties en deze nieuwe operaties bieden mogelijkheden de volgorde van de operaties te optimaliseren. Hiervoor wordt gebruik gemaakt van transformatie regels, bekende heuristieken en de “verdeel en heers” strategie. Bij de optimalisatie behoort ook het uitstellen van operaties die de fragmentatie ongedaan maken. Dit uitstellen biedt de mogelijkheid opeenvolgende operaties parallel te blijven uitvoeren.

Hierna dienen de operaties aan de verschillende processors toegewezen te worden. Hiervoor wordt gebruik gemaakt van allocatie regels. Deze allocatie regels houden de belasting van de verschillende processors bij en wegen verschillende strategieën af. De strategie die de minste stijging in de responstijd oplevert en de belasting gelijkmatig over de processors verdeeld wordt gekozen.

De responstijd wordt verder verminderd door asynchrone communicatie tussen de processors en het optimaal plaatsen van deze communicatie operaties.

Een probleem is echter dat de allocatie grotendeels op basis van schattingen, voor zowel de omvang van tussenresultaten als de processorbelasting, plaatsvindt. Deze schattingen zijn gebaseerd op een uniforme verdeling van de attribuutwaarden in de database. Deze aanname



kan gelden voor de totale gegevensverzameling. Na enkele stappen in het data mining algoritme en daarmee het selecteren van een deelverzameling zal dit echter niet meer zo zijn, aangezien men bij data mining juist op zoek is naar de piekfrequenties van deze waarden. De allocatie van de operaties kan dus beter plaatsvinden op de basis van de werkelijke resultaten van operaties. Het wachten op het beschikbaar zijn van deze resultaten introduceert echter te veel wachttijd op de processors, om dit te voorkomen wordt het toegestaan op basis van een beperkt aantal schattingen vooruit te werken. Dit algoritme wordt dynamische decompositie genoemd.

Een implementatie van dit algoritme toont aan dat hiermee daadwerkelijk de gewenste reductie van de responstijd kan worden behaald. De mate van reductie is echter afhankelijk van verschillende factoren.

Allereerst is daar de gehanteerde fragmentatiemethode. De fragmentatiemethode bepaald namelijk in welke mate er communicatie tussen de processors plaats dient te vinden en in welke mate de belasting over de verschillende processors gelijkmatig kan worden verdeeld. Beide gevolgen hebben invloed op de uiteindelijke responstijd.

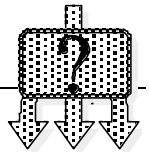
Daarnaast speelt het aantal tuples in de originele database een rol. Bij een klein aantal tuples gaat de belasting als gevolg van de decompositie methode zo’n grote rol spelen dat het parallellisme uiteindelijk geen positief effect meer heeft.

Een volgende factor is het aantal in te zetten processors, hoe meer processors hoe meer reductie er kan worden behaald. Bij een groter aantal processors neemt het aantal tuples per processor af maar de belasting als gevolg van de decompositie methode toe. Hierdoor gaat uiteindelijk dezelfde beperking als bij de voorgaande factor een rol spelen.

Als laatste invloed op de reductie is daar nog de hoeveelheid primair geheugen per processor. Is deze hoeveelheid niet in staat de toegewezen fragmenten te bevatten dan zal de processor gaan swappen naar de harde schijf wat ernstige gevolgen heeft voor de responstijd.

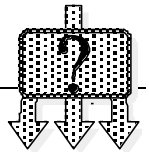
Al deze factoren gezamenlijk beperken de uiteindelijk te behalen reductie van de responstijd met behulp van de ontwikkelde methode.

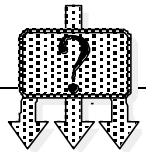
Als conclusie kan er worden gemeld dat de reductie van de responstijd van data mining vragen door parallellisme aanzienlijk kan zijn. Verder onderzoek kan de behaalde resultaten zeker nog bijschaven en de grenzen van de haalbare reductie afbakenen.

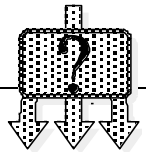


Inhoudsopgave

Hoofdstuk Bladzijde





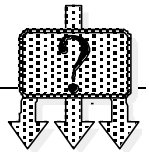


1. Inleiding

Data mining is een reken intensief proces. Niet alleen is er sprake van een enorme hoeveelheid data, maar ook is de probleemruimte die onderzocht moet worden groot. Dit leidt ertoe dat de data mining tool veel vragen aan het database management systeem zal stellen. Het database management systeem zal op zijn beurt een grote hoeveelheid data dienen te doorlopen om het antwoord op de vraag te bepalen.

Dit onderzoek is erop gericht de responstijd van het data mining proces te reduceren. Hierbij richt het onderzoek zich op het reken intensieve gedeelte van het proces, namelijk het beantwoorden van de data mining vraag door het database management systeem. De tijd die aan dit beantwoorden wordt besteed kan worden teruggebracht door gebruik te maken van parallellisme. Dit betekent dat meerdere database management systemen tegelijkertijd werken aan het beantwoorden van de originele data mining vraag. Dit kunnen zij doen door ieder een deel van de vraag te beantwoorden voor het fragment van de database dat zich in hun beheer bevindt. Het probleem dat door dit onderzoek beantwoord dient te worden is: hoe de deelvragen die door de afzonderlijke database management systemen worden behandeld af te leiden uit de originele vraag en deze zo uit te voeren dat er ook daadwerkelijk een reductie van de responstijd plaatsvindt.

De opzet van de scriptie is als volgt. Er wordt begonnen met een korte inleiding over database management systemen (hoofdstuk 2). De theorie die hierin aan de orde komt is van belang voor de twee theoretische onderwerpen die aan de basis van dit onderzoek liggen. Deze theoretische onderwerpen zijn data mining (hoofdstuk 3) en parallelle database management systemen (hoofdstuk 5). Deze theorie wordt dan geïllustreerd met behulp van de praktijksituatie waarin het onderzoek plaatsvond. Voor data mining betreft dit de tool Data Surveyor (hoofdstuk 4) en voor parallelle database management systemen het uitbreidbare parallelle database management systeem Monet (hoofdstuk 6). Daarna wordt op basis van deze theorie en de praktijksituatie de onderzoeksvraag geformuleerd (hoofdstuk 7). Het daaropvolgende hoofdstuk beschrijft de oplossing, oftewel het algoritme (hoofdstuk 8 en 9), dat als antwoord op deze onderzoeksvraag is ontwikkeld en de volgende hoofdstukken beschrijven de implementatie (hoofdstuk 10) en de prestaties (hoofdstuk 11) van deze oplossing. Op basis van deze prestaties worden dan de conclusies over het resultaat van het onderzoek getrokken (hoofdstuk 12). Als afsluitende hoofdstukken worden nog enkele onderwerpen aangegeven die voor verder onderzoek in aanmerking komen (hoofdstuk 13) en een opsomming gegeven van de gehanteerde literatuur (hoofdstuk 14).



2. Database Management Systemen

2.1 Inleiding

In de rest van deze scriptie zullen we ingaan op parallelle data mining. Hierbij zijn twee onderdelen van belang: een data mining applicatie en een parallel te gebruiken database management systeem. Beide onderdelen hebben te maken met de theorie die in de loop van de jaren over database management systemen is ontwikkeld, daarom wordt in dit hoofdstuk eerst een korte inleiding over de werking van database management systemen gegeven. Deze theorie zal goed van pas komen in de volgende hoofdstukken.

2.2 Database systemen

Een database management systeem vormt een onderdeel van een database systeem. Een database systeem bestaat eigenlijk uit twee onderdelen:

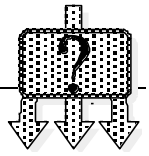
- een database: dit is een verzameling gerelateerde data;
- een database management systeem: een verzameling programma's die het de gebruiker mogelijk maakt een database te creëren, te onderhouden en te ondervragen.

Een gebruiker zet een database systeem op met een bepaald doel. Om de één of andere reden wil hij of zij allerlei feiten over entiteiten in de werkelijke wereld vastleggen om deze later opnieuw op te kunnen vragen. Aangezien een database slechts een beperkt aantal gegevens kan bevatten zal de gebruiker keuzen dienen te maken. Hij of zij moet afwegen welke gegevens van de entiteiten belangrijk zijn voor de vragen die hij of zij in de toekomst wil gaan stellen. Kortom er moet een model, een simplificatie van de werkelijkheid, gemaakt worden. Er zijn verschillende manieren om zo'n model op te bouwen. Men kan bijvoorbeeld een entiteit-relatie model ontwikkelen (zie [Dicker e.a., 1990]) of via normalisatie (zie [Date, 1995]) tot een serie bij elkaar horende gegevens komen. Dit model zal uiteindelijk bestaan uit een serie interessante entiteiten, gewenste gegevens over en relaties tussen deze entiteiten. Daarnaast zullen er bepaalde beperkingen aan de gegevens worden opgelegd, er mogen bijvoorbeeld geen negatieve leeftijden in de database voorkomen.

2.3 Het relationele model

Het database management systeem moet dit model ondersteunen om uiteindelijk de feiten te kunnen opslaan. Vanaf de jaren zestig zijn er verschillende manieren bedacht om de relaties onder te kunnen brengen in een database. Voorbeelden hiervan zijn het hiërarchische model, het netwerk model en het relationele model. Dit derde model, het relationele model, is het meest recent en wordt op wijde schaal toegepast.

Het relationele model heeft een sterk wiskundige onderbouwing in de vorm van relationele algebra. Deze algebra is gebaseerd op de theorie van relaties. Elke entiteit uit de werkelijke wereld wordt vertegenwoordigd door een tabel. Een rij in de tabel wordt een tuple genoemd en bevat de gegevens van één instantie van een entiteit. De kolommen in de tabel bevatten de eigenschappen of attributen van de entiteiten waarvoor in het model is gekozen om ze vast te



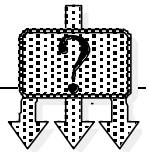
leggen. Een attribuut behoort daarnaast tot een bepaald domein, oftewel elk attribuut heeft een verzameling toegestane waarden.

Elke tuple wordt in een tabel uniek geïdentificeerd door een primaire sleutel. Relaties tussen entiteiten kunnen nu worden vastgelegd door de primaire sleutel van een andere entiteit bij de attributen van de eigen tabel op te nemen, dit wordt een foreign (vreemde) sleutel genoemd.

Als het model is vertaald in een relationeel database schema dient deze aan het database management systeem bekend gemaakt te worden. Hiervoor voorziet het systeem in de Data Definition Language. Via deze taal wordt de data dictionary van het systeem gevuld. In deze data dictionary bevindt zich dus de uiteindelijke beschrijving van het model. Een bijkomend voordeel van de data dictionary is dat de gebruiker wordt afgeschermd van de fysieke layout van de data. Hierdoor kan de database administrator, degene die zorg draagt voor het onderhoud van het database systeem, maatregelen nemen, om bijvoorbeeld de toegangssnelheid tot de data te verhogen, zonder dat de gebruiker zich hiervan bewust is.

De relationele algebra voorziet nu in basisoperaties om de gegevens uit de database op te vragen. De volgende operaties zijn over het algemeen beschikbaar:

- selectie: produceert een nieuwe tabel met alle tuples uit een tabel die voldoen aan de selectie criteria;
- projectie: produceert een nieuwe tabel met maar een paar attributen van de tabel en verwijdert gelijke tuples;
- Cartesisch produkt: produceert een nieuwe tabel die alle attributen van twee tabellen bevat en als tuples alle mogelijke combinaties van tuples uit deze twee tabellen;
- join: produceert een nieuwe tabel met alle combinaties van tuples uit twee tabellen die voldoen aan de join condities;
- vereniging: produceert een nieuwe tabel die alle tuples van twee tabellen bevat, als voorwaarde geldt dat de twee tabellen compatibel zijn, oftewel de attributen en de bijbehorende domeinen dienen gelijk te zijn;
- verschil: produceert een nieuwe tabel die alle tuples van de eerste tabel bevat die niet in de tweede tabel zitten, ook hierbij geldt dat de twee tabellen compatibel moeten zijn;
- doorsnede: produceert een nieuwe tabel die alle tuples bevat die in beide tabellen voorkomen, opnieuw geldt dat de tabellen compatibel met elkaar dienen te zijn.



Sommige van deze operaties zijn door combinaties van de andere samen te stellen. Een database management systeem dient minimaal de selectie, de projectie, de vereniging, het verschil en het Cartesisch produkt te ondersteunen. In ***Fout! Onbekende schakeloptie-instructie.*** zijn de operaties en hun algebraïsche notaties weergegeven.

Operatie	Notatie
selectie	$\sigma_{<\text{selectie condities}>} R$
projectie	$\pi_{<\text{attributen}>} R$
Cartesisch produkt	$R_1 \times R_2$
join	$R_1 \bullet_{<\text{join condities}>} R_2$
vereniging	$R_1 \cup R_2$
verschil	$R_1 - R_2$
doorsnede	$R_1 \cap R_2$

Figuur Fout! Onbekende schakeloptie-instructie. Notatie relationele operatoren

2.4 Het afhandelen van een vraag

De gebruiker stelt zijn vraag niet direct in relationele algebra maar in de Data Manipulation Language. Het database management systeem vertaalt deze taal in de relationele algebra. Voordat deze algebra echter wordt uitgevoerd wordt deze nog geoptimaliseerd. We kunnen via kostenformules namelijk de kostprijs van een uit te voeren vraag berekenen. Via transformatie regels kunnen we de originele vraag herschrijven in een equivalente vraag, die misschien wel goedkoper is. Het aantal equivalente vragen is echter zo groot dat het op zoek gaan naar de aller goedkoopste versie van de vraag niet te doen is, aangezien dit zoeken tijd kost en deze tijd de verkregen optimalisatie weer teniet zou doen. We kunnen echter wel op zoek gaan naar een betere vraagstelling. De transformatie regels die daarbij gebruikt worden maken gebruik van de volgende eigenschappen van de relationele operatoren (zie ook ***Fout! Onbekende schakeloptie-instructie.***):

- commutatieve operaties: de volgorde van de linker of rechter operand van de operatie is niet van belang;
- associatieve operaties: het resultaat van een serie operaties is onafhankelijk van de volgorde waarin ze worden uitgevoerd;
- distributieve operaties: een operatie kan worden verdeeld over een specifieke andere operatie.

Naast het toepassen van deze transformatie regels worden er ook heuristieken toegepast. Als we bijvoorbeeld naar de selectie en projectie operaties kijken dan hebben deze beide de eigenschap dat zij de te behandelen tabel in aantal kolommen of rijen kleiner maken. Het is dus een goed idee deze operaties zo vroeg mogelijk in de vraag uit te voeren en zo de operanden van de volgende operaties kleiner te maken.

Commutatieve operaties

$R_1 \cup R_2 = R_2 \cup R_1$

$R_1 \cap R_2 = R_2 \cap R_1$

$R_1 \bullet R_2 = R_2 \bullet R_1$

Associatieve operaties

$(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$

$(R_1 \cap R_2) \cap R_3 = R_1 \cap (R_2 \cap R_3)$

$(R_1 \bullet R_2) \bullet R_3 = R_1 \bullet (R_2 \bullet R_3)$

Distributieve operaties

$(R_1 \cap R_2) \cup R_3 = (R_1 \cup R_3) \cap (R_2 \cup R_3)$

$(R_1 \cup R_2) \cap R_3 = (R_1 \cap R_3) \cup (R_2 \cap R_3)$

$(R_1 \cap R_2) \bullet R_3 = (R_1 \bullet R_3) \cap (R_2 \bullet R_3)$

$(R_1 \cup R_2) \bullet R_3 = (R_1 \bullet R_3) \cup (R_2 \bullet R_3)$

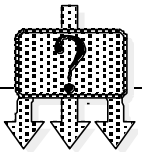
Heuristieken

$\sigma_p(\sigma_q R_1) \equiv \sigma_{p \wedge q} R_1$

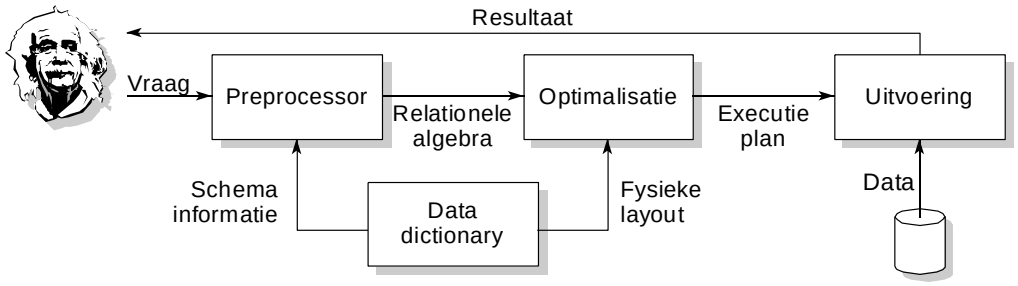
$\pi_p(\pi_q R_1) \equiv \pi_p R_1 \wedge p \subseteq q$

$\sigma_q(R_1 \cup R_2) \equiv (\sigma_q R_1) \cup (\sigma_q R_2)$

Figuur Fout! Onbekende schakeloptie-instructie. Voorbeelden transformatie regels

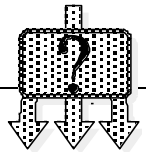


Hiermee is globaal het afhandelen van een vraag aan de database door het database management systeem geschetst. Dit hele traject is afgebeeld in ***Fout! Onbekende schakeloptie-instructie.***.



Figuur Fout! Onbekende schakeloptie-instructie. Het verwerken van een vraag

In het volgende hoofdstuk zal het onderwerp data mining worden besproken. Data mining applicaties maken gebruik van database management systemen om de data waarop zij werken in op te slaan.



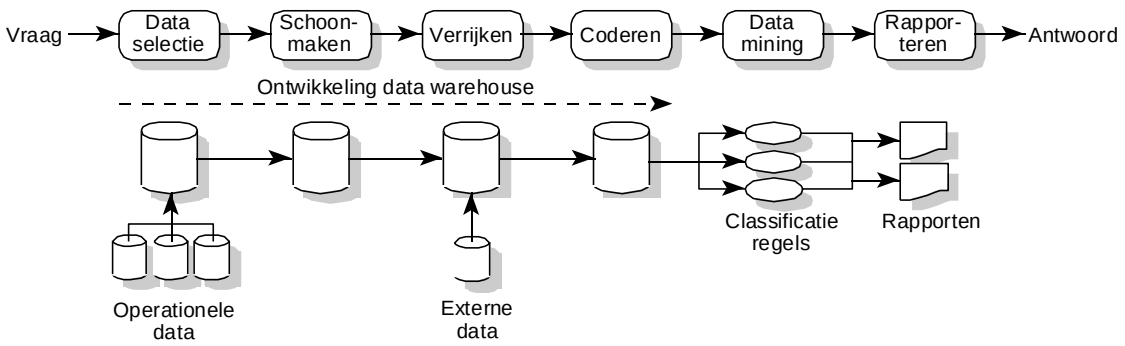
3. Data mining

3.1 Inleiding

Het onderwerp dat in dit hoofdstuk zal worden besproken is data mining. Het Engelse woord mining draagt de betekenis van delven of graven in zich. Bij delven is men op zoek naar iets waardevols: goud of andere schaarse grondstoffen. Bij data mining is dit ook zo. Data mining wordt namelijk als volgt gedefinieerd [Holsheimer e.a., 1994]:

> *Data mining is the search for relationships and global patterns that exist in large databases, but are 'hidden' among vast amounts of data.*

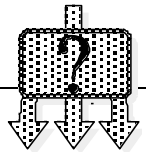
De aarde waarin we aan het spitten zijn is dus een grote verzameling van gegevens, vastgelegd in een database, en de schaarse grondstof waarnaar we op zoek zijn is verborgen kennis, de relaties en patronen in de database. Deze verborgen kennis ligt opgesloten in de overweldigende hoeveelheid gegevens. Deze gegevens zijn over het algemeen afkomstig uit alle hoeken en gaten van een organisatie. Voordat we met data mining kunnen beginnen dienen we deze gegevens eerst te verzamelen. Het totale proces van het verzamelen van de basisgegevens en het vinden van de verborgen of nieuwe kennis wordt Knowledge Discovery in Databases (KDD) genoemd. Data mining is slechts een stap in dit proces, maar wel de stap waarin de nieuwe kennis uiteindelijk wordt gevonden. In ***Fout! Onbekende schakeloptie-instructie.*** worden alle stappen van dit proces weergegeven [Adriaans e.a., 1996].



Figuur Fout! Onbekende schakeloptie-instructie. Het KDD proces

In de afbeelding is KDD als een lineair proces afgebeeld. In de praktijk zal het echter vaak voorkomen dat na het uitvoeren van een bepaalde stap een eerdere stap opnieuw met een iets andere opzet moet worden uitgevoerd. Als er na de data mining stap blijkt dat er niet genoeg gegevens in de database aanwezig zijn om iets relevants op te leveren, zal geprobeerd worden de database verder te verrijken.

De eerste vier stappen hebben tot doel de gegevens en de database zo in te richten dat er ook met behulp van data mining nieuwe en steekhoudende informatie uit kan worden afgeleid. Aangezien het onderwerp van deze scriptie vooral ingaat op de relatie tussen databases en data



mining gaan we ook kort in op deze preparatie van de database. In de volgende paragrafen zullen alle stappen kort de revue passeren.

3.2 De vraag

Voordat we al die moeite doen om de gegevens uit alle hoeken en gaten bij elkaar te halen en te prepareren voor een data mining sessie, moeten we eerst een idee hebben wat realistische vragen zijn. Het probleem dat veel organisaties hebben is dat de informatiehuishouding groeit en groeit, hun informatieverzameling wordt daardoor steeds onoverzichtelijker. Maar zij weten wel dat er allerlei nuttige informatie in die grote verzameling ligt opgesloten. Deze informatie bestaat uit allerlei generalisaties op basis van de individuele feiten die zijn vastgelegd. Met behulp van data mining kunnen we deze generalisaties achterhalen. Deze generalisaties kunnen we dan gebruiken om bepaalde voorspellingen te doen.

Bekende succes verhalen van data mining hebben vaak betrekking op marketing vragen. Marketeers zijn vooral geïnteresseerd in groepen van klanten en kenmerkende vragen zijn bijvoorbeeld:

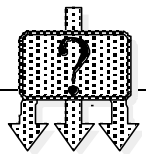
- ik heb een klant met die en die kenmerken, hoe groot is de kans dat deze klant dit produkt zal kopen?
- ik heb dit produkt, wat zijn nu de kenmerken van de kopers van dit produkt?

Deze twee vragen illustreren de twee soorten antwoorden die we met behulp van data mining kunnen verkrijgen.

De eerste vraag heeft betrekking op een voorspelling over het gedrag van een nieuwe klant. Deze vraag kunnen we beantwoorden door te kijken op welke klanten, die we op dit moment in de database hebben, deze nieuwe klant lijkt. We kunnen dan het gemiddelde gedrag van deze nieuwe klant voorspellen aan de hand van de gedragingen van de bekende klanten in die klasse. De tweede vraag heeft betrekking op het achterhalen van een kenmerkende beschrijving, een classificatieregel, van bepaalde groepen (klassen of clusters) klanten in de database.

We zien hier eigenlijk een afhankelijkheid tussen deze twee vragen, de eerste vraag (de voorspelling) is te beantwoorden als de tweede vraag (de classificatieregel) bekend is. Data mining tools verschillen in de manier waarop zij de classificatieregels leren en in welke mate deze regel uiteindelijk inzichtelijk zijn voor de eindgebruiker. Een neuraal netwerk leert bijvoorbeeld wel een classificatieregel, maar hoe deze regel eruit ziet en hoe hij daar aan gekomen is is vaak moeilijk te achterhalen. Neurale netwerken werken in dat geval als een soort black box waar een voorspelling aan kan worden gevraagd. Andere data mining tools, zoals beslissingsbomen leveren wel duidelijke classificatieregels op.

Met andere woorden: we kunnen data mining gebruiken om vragen die op een classificatie probleem neerkomen op te lossen.



3.3 Data selectie

In deze paragraaf wordt de eerste stap beschreven van de prepareren van de data voor een data mining sessie. Het uiteindelijke produkt van deze stappen is een data warehouse (zie [Inmon, 1996]). Wat zo'n data warehouse precies is, welke data ervoor in aanmerking komt en wat het voordeel ervan voor data mining is wordt in deze paragraaf beschreven.

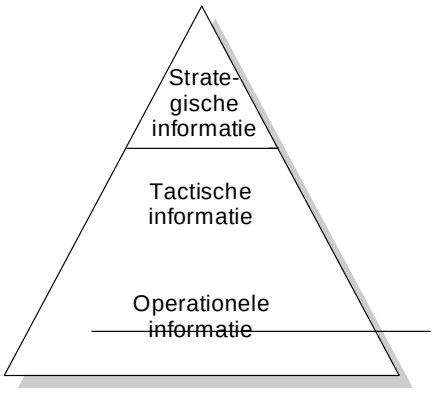
Binnen organisaties treffen we veel verschillende informatiebehoeften aan. Deze informatiebehoeften bestaan eigenlijk uit drie verschillende niveaus met elk hun eigen kenmerken die voorkomen uit het doel waarvoor deze informatie op dat punt in de organisatie wordt verzameld. In ***Fout! Onbekende schakeloptie-instructie.*** worden deze niveaus weergegeven.

Deze informatieniveaus worden als volgt gedefinieerd:

- strategische informatie: dit is informatie waarin de relatie met de omgeving tot uitdrukking komt en die veelal dient voor het nemen van beslissingen die uitwerken op middellange termijn;
- tactische en organisatorische informatie: dit is informatie waarbij de relatie wordt gelegd met de besturing van de interne organisatie en die dient voor het nemen van beslissingen die aangeven hoe de strategische besluiten worden uitgevoerd;
- operationele informatie: dit is informatie waarbij de relatie wordt gelegd met de primaire processen in de organisatie en die dient voor het nemen van beslissingen die samenhangen met het realiseren van tactische besluiten.

Aan de basis van deze piramide vinden we operationele informatie. Dit is eigenlijk alle informatie die benodigd is om het primaire bedrijfsproces en de benodigde ondersteunende processen uit te voeren. Deze informatie wordt gekenmerkt door de volgende eigenschappen:

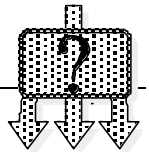
- de informatie is er primair voor de ondersteuning van logistieke processen en de informatiebehoefte is daardoor vrij stabiel;
- de huidige waarde van de informatie is belangrijk en wordt gebruikt voor beslissingen die van dag-tot-dag voorkomen;
- de informatie is te wijzigen;
- de informatie is transactie gedreven en prestatie gevoelig;
- de informatie dient een hoge beschikbaarheid te hebben.



Figuur Fout! Onbekende schakeloptie-instructie. Informatieniveaus

Met deze informatie wordt het dagelijkse werk van de organisatie ondersteund.

Deze informatie vinden we over het algemeen terug in allerlei administratieve systemen. Administratieve systemen hebben tot doel bepaalde informatiebehoeften te ondersteunen. Deze informatiebehoeften hebben betrekking op entiteiten die in de werkelijke wereld een



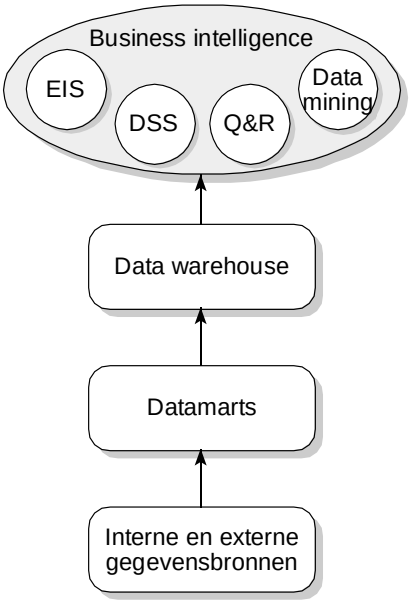
bepaalde relatie met de organisatie hebben. Om informatie over deze entiteiten op te kunnen slaan is er bij het ontwerp van het administratieve systeem een keuze gemaakt van de eigenschappen die deze entiteiten bezitten en de relaties die zij onderling hebben. Deze keuzen leiden tot een model, een simplificatie van de werkelijkheid, en dit model wordt vastgelegd in een zogenaamd database schema. Door deze simplificatie komen we ook aan de clusters of klassen die we in een database met behulp van data mining kunnen onderscheiden, want doordat het aantal meegenomen eigenschappen van elke entiteit beperkt is gaan ze op elkaar lijken en komen op die manier in klassen terecht.

De volgende informatie laag bevat uit deze operationele informatie en hun modellen afgeleide informatie. Deze tactische informatie wordt gebruikt om over korte termijn de organisatie te sturen en is kenmerkend voor afdelingsniveaus. De kenmerken van tactische informatie zijn:

- de informatie bestaat uit zowel afgeleide als historische informatie;
- de behoefte aan informatie ontwikkeld zich continue;
- de informatie wordt niet gewijzigd;
- de informatie wordt via ad-hoc geformuleerde vragen benaderd;
- de informatie is niet zo prestatie gevoelig;
- de informatie heeft een minder hogere beschikbaarheid;
- de informatie handelt over een deel van de organisatie.

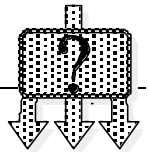
Het laatste niveau, de strategische informatie, verschilt niet zoveel van dit niveau. Het belangrijkste verschil is dat deze informatie niet meer handelt over een deel van de organisatie maar over de complete organisatie. Het termijn waarmee met behulp van deze informatie de organisatie wordt bestuurd is ook langer.

Uit deze behandeling van informatieniveaus kunnen we afleiden dat er eigenlijk twee verschillende soorten databases in een organisatie aanwezig zijn. De eerste soort bevat operationele gegevens. En de tweede soort bevat daaruit afgeleide gegevens. Deze tweede soort wordt ook wel het data warehouse genoemd. Als we voor de afgeleide gegevens ook de tweedeling in tactische en strategische informatie handhaven, dan vinden we nog een tussenniveau de zogenaamde datamarts. Een datamart is een klein(er) data warehouse over een gedeelte van de organisatie. Het samenvoegen van de datamarts leidt weer tot het data warehouse voor de totale organisatie.



Figuur Fout! Onbekende schakeloptie-instructie. Business intelligence

Voor een data mining project is een data warehouse niet verplicht, we kunnen het hele traject van het samenvoegen van operationele data ook eenmalig uitvoeren. Door echter een data warehouse op te bouwen kunnen we naast data mining ook veel andere beslissing ondersteunende systemen inzetten in de organisatie. Een voorbeeld hiervan



is een Executive Information System (EIS) waarmee het strategische niveau in de organisatie allerlei analyses in verschillende maten van gedetailleerdheid kan maken. Er ontstaat dan in de organisatie een informatiehuishouding zoals die is weergegeven in ***Fout! Onbekende schakeloptie-instructie.***. De verwachting is dat al deze toepassingen naar elkaar toe zullen groeien en er een nieuw domein van moderne bestuurlijke informatiesystemen zal ontstaan, in het figuur weergegeven als business intelligence. Business intelligence wordt als volgt gedefinieerd [Hamers, 1996]:

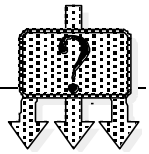
> *Business intelligence is the process of transforming data into information and through discovery transforming that information into knowledge.*

In het geval van data mining vindt deze ontdekking door de computer plaats in tegenstelling tot de andere applicaties die dit ontdekken aan de gebruiker overlaten. Het blijft echter wel de taak van de gebruiker de kennis op haar werkelijke waarde te beoordelen.

Hiervoor hebben we reeds besproken dat de operationele databases werken op basis van een model van de werkelijkheid. Als de werkelijkheid veranderd dan zal dit, bij een goed model, weerspiegeld worden in de database. Deze wijziging in de werkelijke wereld kan tot nieuwe clusteringen in de database leiden en deze komen dan opnieuw via data mining boven water. Data mining is dus een iteratief proces, het is daarom handig een vaste procedure voor de basis gegevens op te zetten. Het onderhouden van een data warehouse is een goede methode om dit te doen.

Kenmerken van de gegevens in een data warehouse ten opzichte van operationele gegevens zijn:

- onderwerp georiënteerd: de gegevens kunnen worden aangepast aan de behoefte van de individuele gebruiker en zijn niet speciaal op één administratief systeem afgestemd;
- het gaat om grote hoeveelheden gegevens: dit is vooral een gevolg van het feit dat er ook historische data wordt bewaard, operationele gegevens hebben over het algemeen alleen betrekking op het heden;
- verschillende opslagmedia: de gegevens komen van overal uit de organisatie en kunnen zich daarom op allerlei opslagmedia bevinden;
- verschillende versies van database schema's: hiervoor is reeds besproken dat een model van de werkelijkheid zich zal ontwikkelen in verloop van de tijd, het database schema dat een weergave van dit model is zal dus ook veranderen, aangezien een data warehouse ook historische gegevens opslaat zullen zich hierin gegevens uit verschillende versies van de database schema's bevinden, oplossingen voor de problemen van het vangen van de tijd in een database komen uit onderzoek over temporele databases (zie [Kersten, 1995] en [McKenzie e.a., 1992]);
- samenvatting en aggregatie van de informatie: de informatiebehoeften in de hogere niveaus van de organisatie zijn anders dan die van het operationele niveau, op zowel de tactische als de strategische niveaus is men niet geïnteresseerd in de individuele data maar in generalisaties en aggregaten hiervan;



- integratie en associatie van informatie uit verschillende bronnen: het data warehouse bevat informatie uit alle hoeken en gaten van de organisatie, daarnaast kunnen, zoals verderop in dit hoofdstuk zal worden besproken, ook allerlei externe informatiebronnen gekoppeld worden aan deze informatie.

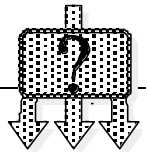
In deze stap van het KDD proces is een belangrijk probleem het integreren van de verschillende interne informatie bronnen. Een groot probleem kan zijn dat binnen verschillende afdelingen van de organisatie andere termen worden gebruikt voor dezelfde (eigenschap van een) entiteit of dat juist één term wordt gebruikt voor verschillende (eigenschappen van) entiteiten. Dit zijn problemen die te maken hebben met de syntax van de verschillende informatiesystemen. Daarnaast kunnen er ook problemen bestaan op het gebied van de semantiek. De berekening van attributen kan op allerlei subtiele manieren verschillen. Zo kan een bepaald bedrag bijvoorbeeld bruto of netto berekend worden. De ene afdeling kan altijd de bruto berekening gebruiken, terwijl een andere altijd de netto berekening gebruikt en beide noemen zij deze eigenschappen bij dezelfde naam. Dit soort verschillen moet expliciet worden gemaakt, anders worden er later conclusies getrokken op basis van attribuutwaarden die verkeerd worden geïnterpreteerd. Een probleem op hetzelfde gebied is dat de sleutels, die binnen de verschillende administratieve systemen zijn gekozen voor de entiteiten, koppelingen tussen verschillende database schema's moeilijk maken doordat ze niet op elkaar aansluiten.

Dit soort verschillen kunnen worden opgevangen door een gezamenlijk gegevensdefinitiebeheer, waardoor de semantiek van de database schema's in de organisatie gelijk wordt getrokken. In dat geval wordt er centraal in de organisatie een soort meta database van definities aangelegd en kunnen veel van dit soort problemen worden ondervangen. Voorbeelden van gegevens die als meta-data dienen te worden opgeslagen zijn:

- wat is de locatie van de data;
- welke data bestaat er;
- wat voor type data is het;
- in wat voor formaat is de data;
- wat is de relatie van de data met de data in andere databases;
- waar komt de data vandaan;
- wie is de eigenaar van de data.

Vanuit zo'n meta database kan eenvoudig een data warehouse worden opgebouwd.

Het ligt aan de fase van integratie van de verschillende administratieve systemen waarin de organisatie zich bevindt hoe makkelijk de meta gegevens zijn te verzamelen en de data ook werkelijk in één database bijeen kan worden gebracht. Aan de ene kant van het spectrum kan de informatiehuishouding bestaan uit allerlei automatiserings eilanden die moeilijk te koppelen zijn, terwijl aan de andere kant het data warehouse al bestaat en zo gebruikt kan worden. In de praktijk zullen beide uitersten niet meer of nog niet zo vaak voorkomen.



Voordat we de gegevens in onze bijeengeraapte verzameling kunnen gaan gebruiken dienen we eerst nog de vervuiling te verwijderen en dat gebeurt in de volgende stap van het KDD proces.

3.4 Schoonmaken

We hebben nu de verschillende operationele databases in de organisatie geïnventariseerd en zijn in staat deze te koppelen tot één verzameling van gegevens, het data warehouse. Deze volgende stap in het KDD proces gaat nu in op een volgend probleem, de vervuiling van databases. Databases in een organisatie verschillen nogal eens in kwaliteit. De vitale gegevens voor de primaire en de ondersteunende activiteiten zullen wel correct zijn, maar allerlei gegevens die buiten de core business vallen hebben een veel grotere kans foutief of niet ingevuld te zijn. Deze niet vitale gegevens kunnen echter juist een onderscheidend kenmerk zijn voor de clusters in de database of kunnen anders het data mining proces danig verstoren. Voordat we nu op onze verzameling gegevens een data mining tool kunnen loslaten dienen we de database daarom eerst schoon te maken. Want ook voor data mining geldt het aloude “garbage in, garbage out”.

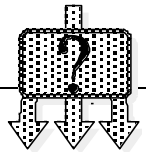
Vervuiling van de database kan op veel manieren, waarvan sommige heel subtiel, plaatsvinden. Enkele manieren zijn:

- missende informatie: de informatie was op het moment van invoeren niet bekend;
- foute informatie: de informatie is verkeerd doorgegeven of ingevoerd;
- default informatie: de gebruiker heeft geen keuze gemaakt en er is een default waarde ingevuld;
- spelfouten: fonetische fouten, verwisselen van letters (bijvoorbeeld: ji i.p.v. ij);
- dubbele informatie: dezelfde entiteit bevindt zich onder andere unieke sleutels vaker in de database.

Er zijn goede hulpmiddelen op de markt te verkrijgen om een deel van deze problemen op te lossen. Spelfouten kunnen met behulp van allerlei spellingssoftware, zoals die ook in tekstverwerkers worden aangetroffen, gecorrigeerd worden. Ook voor het ontdubbelen van databases zijn er hulpmiddelen op de markt.

De eerste drie problemen kunnen gedeeltelijk opgelost worden als deze informatie redundant in de organisatie aanwezig is of extern kan worden aangekocht. Als dit niet mogelijk is hebben we twee keuzen. De eerste keus is deze records uit de database te verwijderen, dit levert geen schade op als deze fouten op een random manier door de database zijn verspreid. Een tweede mogelijkheid is deze informatie van een speciaal kenmerk te voorzien en dit kenmerk mee te nemen in de data mining stap. Dit heeft als voordeel dat men kan ontdekken of er een bepaalde klasse in de database aanwezig is waarbij structureel bepaalde informatie ontbreekt of foutief wordt ingevoerd. Dit soort onderzoeken kunnen gebruikt worden om bijvoorbeeld fraude binnen een organisatie op te sporen.

We dienen goed te beseffen dat de beschikbare volledige en juiste informatie ook bepaald welke antwoorden we op onze data mining vragen kunnen vinden. Als bijvoorbeeld de leeftijd



van onze klanten vaak foutief wordt ingevuld en we deze informatie dus noodgedwongen moeten laten vallen, kunnen we ook nooit meer onze database laten indelen op basis van leeftijden. We kunnen hierdoor net het discriminerende kenmerk van bepaalde klassen in deze stap kwijt raken. Gelukkig is er nog veel informatie op de markt te koop om onze database mee te verrijken. Deze stap wordt in de volgende paragraaf behandeld.

3.5 Verrijken

Met de vorige stappen hebben we uit alle hoeken en gaten van onze organisatie gegevens bij elkaar gehaald en de vervuiling eruit verwijderd. In deze stap kunnen we nu nog meer informatie aan onze verzameling toevoegen.

In veel landen is enorm veel informatie te koop. Zo zijn er allerlei demografische gegevens over klanten te koop en ook allerlei gegevens over bedrijven. Bronnen voor dit soort informatie zijn:

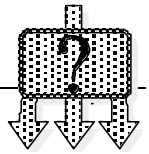
- telecommunicatie bedrijven (de PTT);
- kamer van koophandel;
- allerlei bedrijven die zelf informatie in handen hebben en misschien willen ruilen voor de informatie die de eigen organisatie heeft;
- organisaties die het koopgedrag van klanten registreren met behulp van klantenkaarten, credit cards en dergelijke.

De problemen die spelen bij het samenvoegen van de eigen gegevens en de externe gegevens zijn dezelfde als die reeds bij de data selectie stap zijn aangegeven. Voordat we nu uit deze verzameling van interne en externe gegevens nieuwe kennis kunnen afleiden dienen we de database nog te prepareren zodat de data mining algoritmes makkelijk met de gegevens overweg kunnen.

3.6 Coderen

Bij de data selectie en het verrijken van de gegevensverzameling is reeds aan de orde gekomen dat we gegevens uit verschillende bronnen met elkaar kunnen koppelen door een gezamenlijke sleutel. Deze onderlinge relatie tussen twee of meerdere gegevensverzamelingen kan echter andere interessante afhankelijkheden verhullen. Daarom vindt data mining plaats op de universele relatie. In de universele relatie zijn alle attributen uit de verschillende gegevensverzamelingen samengebracht in één tabel. Hierdoor zijn alle mogelijke afhankelijkheden direct beschikbaar voor het data mining algoritme.

Tot nu toe hebben we allerlei gedetailleerde informatie uit de operationele databases gehandhaafd. Deze details hadden we nodig om de interne en de externe gegevensbronnen onderling te kunnen koppelen en de universele relatie te kunnen bepalen. Sommige van de gegevens in de universele relatie zijn echter veel te gedetailleerd om er generalisaties uit af te kunnen leiden. Een marketeer is niet geïnteresseerd in het aankoopgedrag van een enkel huishouden maar in het aankoopgedrag van een bepaalde regio.



Dit soort afwegingen kunnen gelden voor veel meer eigenschappen die in de database zijn opgeslagen. Enkele nuttige transformaties zijn:

- adressen naar regio's: bijvoorbeeld naar postcode gebied, provincie of land;
- data naar jaren of maanden: bijvoorbeeld geboortedatum naar leeftijd;
- bedragen naar intervallen: bijvoorbeeld inkomen op de cent nauwkeurig naar intervallen van 1000 gulden;
- type velden naar meerdere booleaanse velden: bijvoorbeeld type rekening naar spaarrekening en betaalrekening.

Via deze transformaties generaliseren we een deel van de gedetailleerde informatie. We kunnen nu een bepaalde klasse aanwijzen (bijvoorbeeld alle klanten met een spaarrekening) en de data mining tool voor deze klasse een classificatieregel laten afleiden. Dit is dan ook de volgende stap die we zullen gaan bespreken.

3.7 Data mining

3.7.1 Inleiding

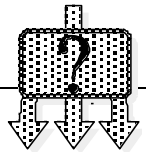
Nu komen we dan uiteindelijk bij de stap die de nieuwe informatie uit de gegevensverzameling moet gaan afleiden. Nieuwe informatie bevindt zich op verschillende niveaus in deze gegevensverzameling. Voor elk van deze niveaus zijn weer andere hulpmiddelen beschikbaar. De niveaus die worden onderscheiden zijn:

1. ondiepe of oppervlakkige informatie: te ontdekken met behulp van SQL;
2. multi-dimensionele informatie: te ontdekken met OLAP;
3. verborgen informatie: te ontdekken met data mining;
4. diepe informatie: alleen te ontdekken met aanwijzingen.

Tot deze laatste soort informatie hoort bijvoorbeeld een versleuteld bericht. We kunnen de informatie die in dit bericht ligt opgesloten alleen achterhalen als we de sleutel weten, als er tenminste van een goede versleutelingstechniek gebruik is gemaakt.

Zo'n 80% van de interessante informatie bevindt zich echter niet al te diep in de database en kunnen we met behulp van Structured Query Language (SQL) boven de bodem krijgen. We kunnen denken aan allerlei interessante aggregaten, zoals het rekenkundige gemiddelde. De gemiddelde leeftijd van de lezers van een tijdschrift tezamen met de spreidingsbreedte geeft al heel wat informatie over een bepaalde klasse in de database. Deze informatie is zonder al te veel moeite uit de database te halen en de classificatieregels en voorspellingen die we met behulp van data mining boven water krijgen moeten dan ook beter scoren dan deze zogenaamde naïeve voorspellingen.

De volgende laag informatie wordt verkregen door gegevens tegen elkaar af te zetten in een multi-dimensionale ruimte. Een voorbeeld voor drie dimensies is: de verkopen in de verschillende regio's, afgezet tegen de leeftijdsgroepen en het geslacht. Dit soort vragen worden ondersteund door On-Line Analytical Processing (OLAP) en biedt de gebruiker de mogelijkheid allerlei dimensies tegenover elkaar te zetten. De gebruiker dient zelf echter de



verkregen gegevens te interpreteren. OLAP applicaties bieden daarvoor allerlei visualisatie methoden om dit makkelijker te maken. Als we de computer de interpretatie willen laten doen begeven we ons op het terrein van het machine leren. Veel van de algoritmen, die we hierna voor data mining zullen bespreken, zijn dan ook ontstaan in dit vakgebied.

Deze eerste twee niveaus van informatie helpen ons te achterhalen in welke stukken van de database we interessante verborgen informatie kunnen aantreffen. Het verder analyseren van deze interessante gebieden voor het opdiepen dan informatie van het derde niveau gebeurt dan met behulp van data mining tools.

De leer methodiek die bij data mining gebruikt wordt is inductief leren. Bij inductief leren probeert men op basis van een serie gegeven feiten een algemene uitspraak te doen. Een voorbeeld hiervan is:

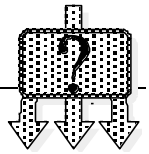
Gegeven: Ernie is een kind
Gegeven: Ernie kijkt naar Sesamstraat
Gegeven: Bert is een kind
Gegeven: Bert kijkt naar Sesamstraat
Inductie: Sesamstraat wordt door kinderen bekeken

Dit leren van een generalisatie kunnen we zien als het comprimeren van de gegevensverzameling. In plaats van al de gegevens te onthouden hoeven alleen maar de klasse van de entiteiten te onthouden en met behulp van de gevonden classificatieregel(s) kunnen we de overige eigenschappen invullen. Dit is nu precies wat we bij data mining willen: de database in clusters opdelen en voor elke cluster een classificatieregel vinden. Het zal echter heel moeilijk zijn voor elke cluster één classificatieregel te vinden die alle entiteiten in die klasse voor 100% beschrijft. We hebben dus dezelfde situatie als bij de compressie van beelden. Als we de GIF compressie voor een beeld gebruiken wordt de dataset gecomprimeerd zonder detail informatie te verliezen. Als we de JPEG compressie gebruiken is de compressie veel groter maar zullen we wel detail informatie verliezen. Het ligt nu aan de toepassing en de mate van detail verlies of dit verlies acceptabel is. Net als bij beelden kunnen we bij data mining er voor kiezen op zoek te gaan naar classificatieregels die voldoende (maar niet 100%) statistische dekking in de database hebben om nieuwe gevallen te voorspellen.

Data mining is geen enkele techniek, maar elke techniek dat tot een beter inzicht in de verzameling gegevens kan leiden kan worden ingezet. Vandaar dat zeer verschillende technieken als data mining tools worden aangeboden, enkele daarvan zijn:

- *k*-naaste buren;
- beslissingsbomen;
- associatie regels;
- neurale netwerken;
- genetische algoritmen.

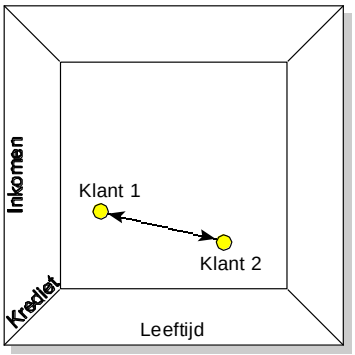
Zoals reeds is aangeven werken sommige methoden beter bij voorspellingen en leveren andere duidelijkere classificatieregels op. Voor zowel *k*-naaste buren en neurale netwerken geldt dat



deze werken als een black box waarin de classificatieregel zit verborgen en een voorspelling uitkomt. De andere methoden leveren wel een duidelijke classificatieregel op, die daarna kan worden gebruikt om een voorspelling te doen. We zullen al deze methoden even kort aanstippen.

3.7.2 *k*-naaste buren

Allereerst de *k*-naaste buren. Met behulp van deze methode kunnen we voor een nieuwe klant proberen zijn gedrag te voorspellen door te kijken naar de manier waarop zijn buren hebben gereageerd. De *k* geeft aan naar hoeveel van deze buren we kijken om de reactie van de nieuweling te voorspellen. Een belangrijk concept bij deze methode is dus, wie is een buur en wie niet. Buren worden gevonden door de instanties weer te geven in een *n*-dimensionale ruimte. Het aantal dimensies wordt bepaald door het aantal eigenschappen dat we bij de clustering gebruiken. Hierdoor kunnen we een afstandsmaat, bijvoorbeeld de euclidische afstand, gebruiken om de afstand tussen twee buren weer te geven. Het is dan wel belangrijk dat de eigenschappen worden genormaliseerd zodat ze elk even zwaar meetellen in de berekening van de afstand, dit wordt geïllustreerd in ***Fout! Onbekende schakeloptie-instructie.***. Via deze afstandsmaat kunnen nu de *k* buren worden gevonden en de gemiddelde reactie worden bepaald.



Operationele gegevens:

	Leeftijd	Inkomen	Krediet
Klant 1	32	40.000	10.000
Klant 2	65	30.000	2.000

Genormaliseerd:

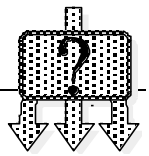
	Leeftijd	Inkomen	Krediet
Klant 1	0,32	0,4	0,1
Klant 2	0,65	0,3	0,02
Verschil	0.33	0,1	0,08

Afstand:

$\sqrt{}$ ($0{,}33^2 + 0{,}1^2 + 0{,}08^2$) = 0,35

Figuur Fout! Onbekende schakeloptie-instructie. De afstand tussen twee buren

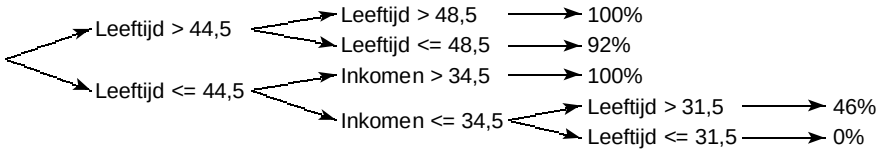
Een probleem bij deze methode is dat bij een oplopend aantal dimensies de *n*-dimensionale ruimte steeds leger wordt en de afstanden tussen buren steeds meer gelijk. Een *k*-naaste buren algoritme scoort het beste op een kleine aantal dimensies. Er zijn gelukkig methoden om grotere aantal dimensies te transformeren naar een kleiner aantal dimensies. Zoals reeds eerder is aangegeven leent *k*-naaste buren zich het beste voor het doen van voorspellingen. Je stopt een nieuwe klant in de black box en er komt een voorspelling op basis van het gedrag van een *k* aantal buren uit. Hoe de classificatie regels eruit zien blijft echter verborgen in de black box. De volgende methode, beslissings bomen, maakt deze regels wel expliciet.



3.7.3 Beslissingsbomen

De volgende methode zijn beslissingsbomen. Deze bomen geven eigenlijk een pad aan dat we kunnen volgen om uiteindelijk een nieuwe klant te classificeren. Het uitgangspunt hierbij is dat er een bepaalde volgorde van belangrijkheid in de verschillende eigenschappen te vinden is. Er wordt begonnen met de eigenschap die de meeste informatie geeft. Bij een lezersdatabase kan dit bijvoorbeeld de leeftijd zijn, de meeste lezers van Donald Duck zijn jonger dan 13 jaar. Hiermee hebben we de database dan in twee stukken gedeeld, één jonger dan 13 en één ouder. Elk van deze stukken kunnen we met ander eigenschap weer verder onderverdelen. Nieuwe vertakkingen worden alleen in de boom opgenomen als zij leiden tot een betere specificatie van een deel van de database.

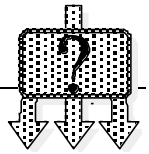
In ***Fout! Onbekende schakeloptie-instructie.*** wordt een beslissingsboom weergegeven voor een onderzoek naar de lezers van een auto magazine [Adriaans e.a., 1996]. Bij de eerste knoop is het onderscheidend vermogen van een drempelwaarde van 44,5 voor leeftijd 99% en 38%. Dit betekent dat in de database 99% van de lezers met een leeftijd hoger dan 44,5 het auto magazine niet leest. Bij de volgende knoop is er verbetering mogelijk want een leeftijdsgrens van 48,5 jaar leidt tot een score van 100%, oftewel in de leeftijdsgroep ouder dan 48,5 leest niemand het auto magazine. Gelijksoortige overwegingen gelden voor de rest van de takken in de boom. Het pad “Leeftijd <= 44,5 en Inkomen <= 34,5 en Leeftijd <= 31,5” leidt tot een dekking van 100% voor lezers van het auto magazine.



Figuur Fout! Onbekende schakeloptie-instructie. Beslissingsboom

Het probleem bij deze methode is dat het aantal classificatieregels dat mogelijk is exponentieel toeneemt met het aantal eigenschappen. Slechts een klein aantal van deze regels is statistisch relevant in die zin dat ze goed discrimineren en dat ze genoeg dekking in de database hebben. We kunnen daarom aan een regel een kwaliteitswaarde hangen, bijvoorbeeld de percentages die in de voorbeeld boom zijn gebruikt. Hierdoor ontstaat er als het ware een kwaliteitslandschap met pieken en dalen. Door nu een slimme zoektechniek, zoals hill climbing of simulated annealing, te gebruiken kunnen we op zoek naar de hoogste piek, oftewel de classificatie regel met de hoogste dekking en de beste discriminerende eigenschap, zonder alle regels te genereren.

Opnieuw speelt hierbij een buren probleem. Wie is in welke richting de buur van een bepaalde classificatieregel? De methode die de beslissingsboom uit het voorbeeld heeft genereert bouwt een binaire boom op, andere methoden staan meer dan twee zijtakken toe per knoop. Een knoop kan ook alleen zijtakken genereren op basis van één eigenschap of op basis van meerdere eigenschappen.



Deze algoritmes verschillen in drie onderdelen: de kwaliteitsfunctie, de zoekmethode en wie is een buur. Verderop in deze scriptie zal er een methode om op zoek te gaan naar beslissingsbomen verder worden uitgewerkt.

Voor informatie over beslissingsbomen zie [Quinlan, 1986].

3.7.4 Associatieregels

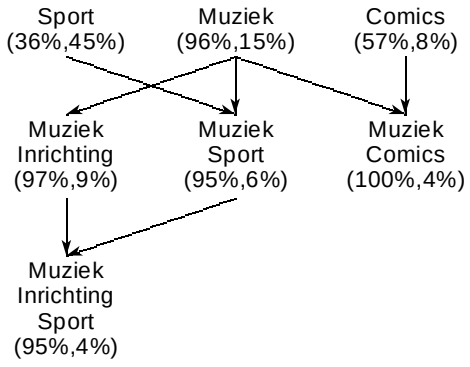
Associatieregels worden gebruikt om veel samen voorkomende waarden van eigenschappen te zoeken. Stel dat we bij een kassa van een supermarkt opslaan welke artikelen zich samen in één winkelkarretje bevonden. Via data mining kunnen we nu op zoek gaan naar de complementaire goederen, bijvoorbeeld 60% van de klanten die een pak koffie kopen koopt ook een fles koffiemelk. Dit soort regels noemen we associatieregels. Associatieregels werken alleen op binaire of booleaanse eigenschappen, zoals een winkelkarretje bevat een pak koffie of niet. De database moet dus in de coderings stap opgezet zijn zoals is weergegeven in ***Fout! Onbekende schakeloptie-instructie.***.

Koffie	Koffiemelk	Koekjes	Suiker	Wasmiddel
ja	nee	ja	nee	ja
nee	nee	ja	nee	ja
ja	ja	ja	ja	nee

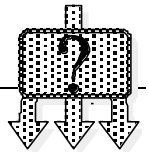
Figuur Fout! Onbekende schakeloptie-instructie. Winkelkarretjes bestand

In een gegevensverzameling kunnen we echter, net als bij de beslissingsbomen, veel associatie regels afleiden. Opnieuw hebben we dus een soort maat nodig voor de kwaliteit van een associatie regel. We kunnen twee maten gebruiken. De eerste is de dekking in de database, oftewel welk percentage van de winkelwagentjes bevatten zowel koffie als koffiemelk. De tweede maat geeft een indruk van de betrouwbaarheid van de regel: welk percentage van de winkelwagentjes die koffie bevatten, bevatten ook koffiemelk. Deze twee maten samen geven een indruk van het statistische belang van de associatieregel.

In ***Fout! Onbekende schakeloptie-instructie.*** staan de associatieregels voor een onderzoek naar de lezers van een aantal tijdschriften. De associatieregels geven aan welke lezers een abonnement hebben op meerdere tijdschriften. In de methode die is gebruikt moet er een doel worden aangegeven, in dit geval is dat een tijdschrift voor auto's. Onder de regels staan zowel de dekking als het vertrouwen vermeld. De associatieregel "Muziek, Sport" heeft een dekking van 6%, dus van de lezers in de database heeft 6% een abonnement op beide bladen en het auto tijdschrift. De betrouwbaarheid van deze regel is 95%, oftewel 95% van de lezers die een abonnement hebben op het muziek en een sport tijdschrift hebben ook een abonnement op het auto tijdschrift. Met behulp van de twee maten kan bepaald worden welke



Figuur Fout! Onbekende schakeloptie-instructie. Associatie regels voor een auto tijdschrift



associatieregel interessant is om verder uit te diepen en hebben we ongeveer dezelfde aanpak als bij de beslissingsbomen.

Voor informatie over associatieregels zie [Agrawal e.a., 1993].

3.7.5 Neurale netwerken

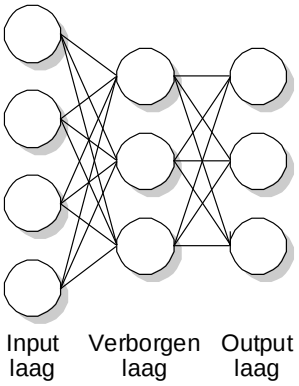
Neurale netwerken kunnen ook helpen bij classificatie problemen. In dat geval werken ze net als de *k*-naaste buren methode als een black box waaruit een voorspelling komt. Neurale netwerken zijn er in verschillende soorten en maten. In deze behandeling zullen we kort ingaan op back propagation en Kohonen netwerken. Een netwerk bestaat uit twee onderdelen:

- neuronen, georganiseerd in lagen;
- gewogen verbindingen tussen de neuronen in verschillende lagen.

In ***Fout! Onbekende schakeloptie-instructie.*** wordt dit geïllustreerd. Een netwerk heeft één input laag waar de eigenschappen worden aangeboden en één output laag waar de klasse wordt aangegeven. In het netwerk bevinden zich één of meer verborgen lagen. Als het netwerk begint met het aanleren van een classificatie regel zullen de gewichten van de verbindingen met willekeurige waarden zijn geïnitialiseerd. De neuronen bevatten een activatie functie die bepaald welke waarde de neuron produceert. De input functie van deze activatie functie is de som van de waarden die de neuron via de gewogen connecties binnenkrijgt. Een simpele activatie functie is bijvoorbeeld de “sign”-functie:

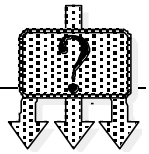
$$sign(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

De input propageert via deze berekening door naar de output neuronen. Door nu deze output te vergelijken met de gewenste output komen we de fout te weten. Deze fout wordt nu terug gepropageerd door het netwerk en gebruikt om de gewichten aan te passen. Door nu een hele serie input en output paren aan te bieden kunnen we het netwerk een classificatie regel leren. Het leren is klaar als het netwerk niet meer wijzigt en dan kunnen we het netwerk gebruiken als de black box voor de voorspelling. Bij een voorspelling bieden we alleen de input aan en krijgen de output volgens de aangeleerde classificatie functie.



Figuur Fout! Onbekende schakeloptie-instructie. Neuraal netwerk

Een ander soort neurale netwerken dat gebruikt kan worden om een classificatie functie te leren zijn de Kohonen netwerken. In dit geval zijn de neuronen verdeeld over een *n* dimensionale ruimte en verbonden met een beperkt aantal buren. Als er nu een input wordt aangeboden dan zal de dichtstbijzijnde neuron afgaan. Van dit neuron worden dan gewichten zo aangepast dat hij naar de input toe wordt getrokken, daarbij trekt hij voor een deel zijn buren mee. Door nu een serie input paren aan te bieden gaan de neuronen zich in bepaalde clusters bevinden. Als het netwerk stabiel is geworden kan men hier ook nieuwe input aanbieden en kijken welke clusters er reageren.

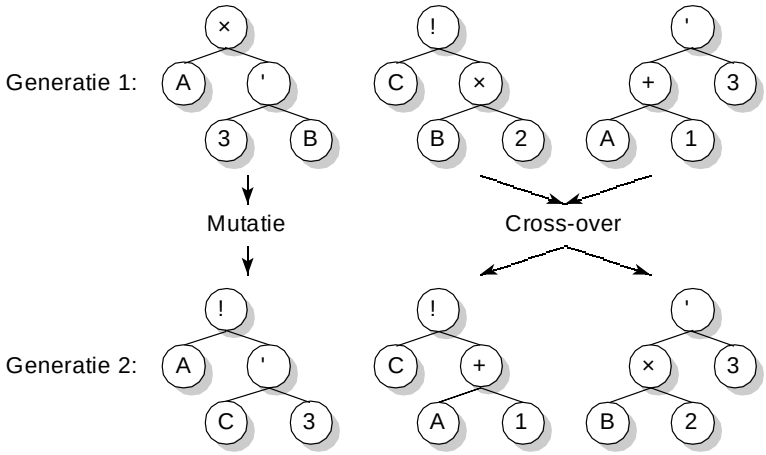


Voor algemene informatie over neurale netwerken zie [Kröse e.a., 1995] en [Hertz, 1991].

3.7.6 Genetische algoritmes

Genetische algoritmes werken met behulp van strings. Deze strings beschrijven een mogelijke classificatieregel. Met behulp van een fitness functie kan worden bepaald wat de kwaliteit is van de classificatieregel die door de string wordt beschreven. Via deze fitness functie wordt bepaald welke strings in aanmerking komen om verder te evolueren. Dit evolueren gebeurt met behulp van mutatie en cross-over operaties. Op een gegeven moment zullen de strings convergeren naar een optimum en hebben we een optimale classificatie regel.

Als voorbeeld kunnen we op zoek gaan naar een bepaalde functie die twee klassen scheidt. De onderdelen van de string zijn dan constanten, variabelen en operatoren. Deze string kunnen we visualiseren als een operator boom. De kwaliteit van deze string kan bijvoorbeeld worden gevangen in de kleinste kwadraten methode (de som van de kwadratische afstand van de alle punten tot de functie). De string kan gemuteerd worden door willekeurig de operatoren te verwisselen door andere operatoren en de variabelen en constanten onderling te verwisselen. Cross-over operaties zouden bestaan aan het onderling verwisselen van zijtakken van de operator bomen. Via deze operaties zouden dan weer nieuwe strings ontstaan waarvan de kwaliteit dan weer kan worden bepaald. In ***Fout! Onbekende schakeloptie-instructie.*** worden de operaties geïllustreerd.

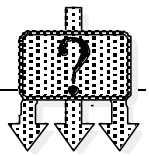


Figuur Fout! Onbekende schakeloptie-instructie. Mutatie en cross-over operaties

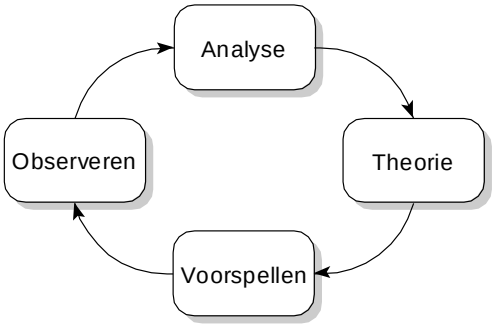
Voor algemene informatie over genetische algoritmes zie [Michalewicz, 1994].

3.8 Rapporteren

Het rapporteren van de uitkomst van de verschillende methoden kan zowel tekstueel als grafisch gebeuren. Grafische representatie kan op de manieren plaatsvinden zoals die in de figuren in dit hoofdstuk zijn weergegeven, daarnaast zijn er ook nog allerlei grafiek vormen die gebruikt kunnen worden om de informatie inzichtelijk te maken. Ook bestaat het rapporteren uit een voorziening voor het gebruik van de gevonden classificatieregels bij het



voorspelen. Bij *k*-naaste buren en de neurale netwerken zal dit bestaan uit een programma dat de rol van black box op zich neemt. De andere methoden leveren leesbare regels op die eventueel zelf door de gebruiker kunnen worden toegepast of in een programma kunnen worden ingebouwd.



Figuur Fout! Onbekende schakeloptie-instructie. Iteratief proces

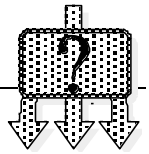
3.9 Het antwoord

Zoals reeds eerder is aangegeven moeten we de antwoorden die we uit een data mining sessie krijgen wel met een kritisch oog beschouwen. Enkele redenen hiervoor zijn:

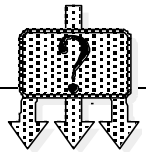
- de basisgegevens komen uit de modellen van de werkelijkheid die voor de operationele toepassingen zijn gemaakt, aangezien een model altijd een simplificatie van de werkelijkheid is betekent dit dat er eigenschappen missen, deze missende eigenschappen kunnen een heel ander beeld opleveren;
- de generalisaties worden gemaakt op basis van de gegevens in de database, deze database bevat niet de totale populatie van de gemodelleerde entiteiten, daardoor kan de data mining tool bijvoorbeeld tot de generalisatie komen dat alle zwanen wit zijn, in het werkelijke wereld komen echter ook zwarte zwanen voor;
- de situatie in de werkelijke wereld is niet stabiel maar veranderd voortdurend en dit betekent dat generalisaties die vroeger golden op dit moment niet meer hoeven te gelden.

Al deze redenen pleiten ervoor dat data mining een iteratief proces wordt volgens de iteratie in ***Fout! Onbekende schakeloptie-instructie.***. De analyse stap hebben we nu verricht door middel van de data mining tool. Deze analyse stap heeft een theorie opgeleverd, die we nu gaan gebruiken om voorspellingen te doen. De theorie heeft echter slechts een beperkte geldigheidsduur en door de werkelijke wereld te observeren kunnen we het moment bepalen dat de theorie niet meer geldt of er achter komen dat hij nooit heeft geklopt en we een nieuwe moeten gaan ontwikkelen.

Dit observeren heeft te maken met een principe dat door de filosoof Karl Popper werd ontdekt. Dit principe geeft weer dat een generalisatie nooit door een eindig aantal waarnemingen kan worden geverifieerd, maar wel door één waarneming gefalsifieerd. Kortom hoeveel waarnemingen we ook in onze database hebben we kunnen nooit zeker weten of de theorie die we op basis van deze gegevensverzameling hebben opgesteld voor de hele populatie geldt. We hoeven echter maar één waarneming te hebben die niet aan de theorie beantwoord om de theorie op losse schroeven te zetten. Bij het observeren is het dus de bedoeling te letten op deze waarnemingen die tegen de theorie ingaan. Het moment om naar een nieuwe generalisatie te delven is daar wanneer de huidige theorie zijn statistische dekking verliest.



Hiermee zijn alle stappen uit het KDD proces besproken, inclusief de data mining stap. In het volgende hoofdstuk zal een specifieke data mining tool, Data Surveyor, worden besproken.



4. Data Surveyor

4.1 Inleiding

In dit hoofdstuk wordt een beschrijving gegeven van de data mining tool die in de praktijksituatie van deze scriptie een rol speelt. Deze tool, Data Surveyor, hanteert een algemeen raamwerk voor data mining algoritmes. De huidige implementatie van dit raamwerk ondersteunt het genereren van beslissingsbomen. Dit hoofdstuk begint met een paragraaf over dit achterliggende raamwerk en vervolgt dan met een beschrijving van een specifieke invulling van dit raamwerk voor het genereren van de beslissingsboom. In de laatste paragraaf zal de relatie tussen Data Surveyor en het ondersteunende database management systeem, wat een belangrijke rol speelt in het vervolg van deze scriptie, duidelijk worden.

4.2 Het algemene raamwerk

Het algemene raamwerk beschrijft de samenhang tussen de verschillende onderdelen van een data mining algoritme. Zoals reeds is beschreven richten deze algoritmes zich op het vinden van classificatieregels van interessante deelverzamelingen in de gegevensverzameling. Deze interessante deelverzamelingen kunnen worden beschreven door twee onderdelen:

- de syntax: een beschrijving van de vorm van de interessante deelverzameling (bijvoorbeeld een pad door een beslissingsboom);
- de semantiek: een kwaliteitsfunctie die aangeeft hoe interessant de deelverzameling werkelijk is (bijvoorbeeld hoeveel dekking heeft de classificatieregel in de database).

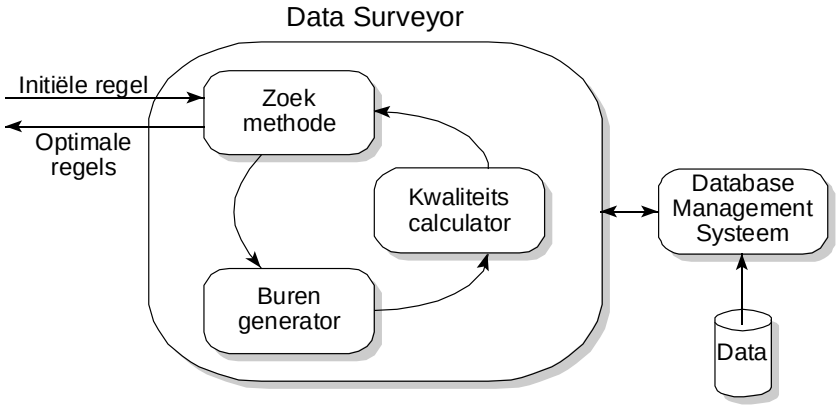
Nu we een definitie hebben van interessante deelverzamelingen kunnen we op zoek gaan naar de interessantste deelverzameling.

Het zoeken vindt plaats met behulp van een zoekmethode. Zoekmethodes springen van de ene syntactische beschrijving van een deelverzameling in de zoekruimte naar de volgende. Dit springen gebeurt op basis van de semantische beschrijving van de deelverzameling. Het zoekalgoritme tast als het ware de omgeving van de actieve deelverzameling af op zoek naar een buur die een betere kwaliteitswaarde oplevert. Dit betekent dat we een burenbegrip nodig hebben, oftewel op basis van de beschrijving van de huidige deelverzameling moet een nieuwe aanverwante beschrijving kunnen worden gegenereerd.

Hiermee zijn de verschillende onderdelen van het raamwerk de revue gepasseerd, namelijk:

- een generator voor een syntactische beschrijving van een deelverzameling;
- een kwaliteitscalculator voor de semantische waarde van een deelverzameling;
- een zoekmethode om op zoek te gaan naar de interessantste deelverzameling.

Deze onderdelen vinden we terug in de in ***Fout! Onbekende schakeloptie-instructie.*** afgebeelde architectuur van Data Surveyor. Het systeem wordt gevoed met een initiële beschrijving, dit is het doel of de klasse waarvoor de tool opzoek gaat naar de classificatieregels. Om nieuwe beschrijvingen te kunnen genereren en de kwaliteit van deze te kunnen berekenen heeft het systeem ook toegang nodig tot de onderliggende gegevensverzameling.



Figuur Fout! Onbekende schakeloptie-instructie. Architectuur Data Surveyor

In de volgende paragraaf wordt de specifieke invulling van dit raamwerk voor het genereren van beslissingsbomen besproken.

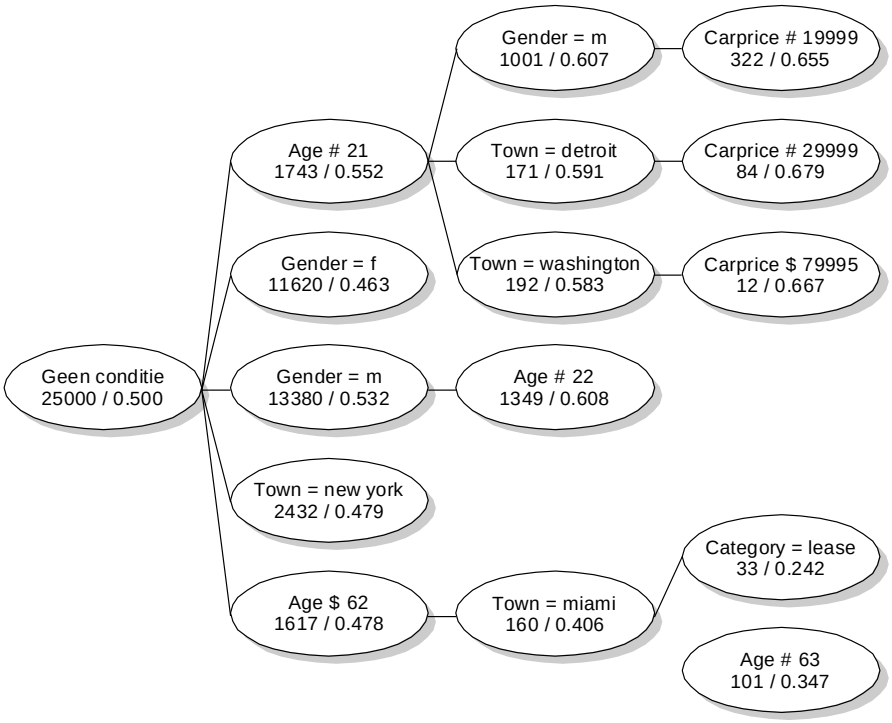
4.3 Het genereren van beslissingsbomen

In de vorige paragraaf is een algemeen raamwerk voor data mining algoritmes besproken. Dit raamwerk wordt binnen Data Surveyor gebruikt om beslissingsbomen te genereren. De drie onderdelen van het raamwerk dienen dus voor beslissingsbomen geïmplementeerd te zijn.

Allereerst de syntax, oftewel de beschrijving van de deelverzamelingen. Bij beslissingsbomen is dit de beslissingsboom zelf. Deze boom bevat een aantal classificatieregels: elk pad door de boom is namelijk zo'n regel. Op basis van een gegevensverzameling kunnen echter verschillende bomen worden gegenereerd, welke boom is afhankelijk van het burenbegrip. In de bespreking van beslissingsbomen in het voorgaande hoofdstuk is in ***Fout! Onbekende schakeloptie-instructie.*** een boom weergegeven die in elk knooppunt slechts twee keuzen biedt en deze gaan beide over hetzelfde attribuut. In Data Surveyor biedt een knooppunt toegang tot meerdere nieuwe knooppunten en deze knopen kunnen over meerdere attributen gaan. Als beperking geldt dat elk attribuut maar één keer in een regel voorkomt.

Het volgende onderdeel was de semantiek van deze beschrijvingen. De waarde van een beschrijving wordt uitgedrukt in een kwaliteitswaarde en de waarde wordt berekent door een kwaliteitsfunctie. Als we naar de boom kijken dan bevat elk knooppunt een selectiecriterium op de onderliggende gegevensverzameling. Een classificatieregel is, in dit geval, dus niets anders dan de conjunctie van selectiecriteria op verschillende attributen. Verder hebben we bij het opstarten van het algoritme een initiële regel opgegeven, deze regel geeft het doel van de zoektocht aan namelijk een classificatieregel of een verzameling classificatieregels voor een bepaalde klasse in de gegevensverzameling. De kwaliteit van een gevonden regel kan nu worden uitgedrukt in de volgende kwaliteitsfunctie *Q*:

$$Q(R) = \frac{\left|\sigma_D(P)\right|}{\left|\sigma_D(P)\right| + \left|\sigma_D(N)\right|}$$



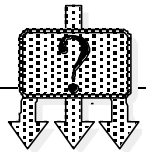
Figuur Fout! Onbekende schakeloptie-instructie. Een 5 × 3 beslissingsboom voor een verzekeringsmaatschappij

Hierbij is *R* de gevonden regel. *P* bevat de tuples die behoren tot de klasse en *N* degenen die niet de klasse behoren. *D* is het de conjunctie van de selectiecriteria van regel *R* en σ geeft de selectieoperatie aan. De kwaliteit is dus de verhouding tussen het aantal tuples dat aan de regel voldoet en tot de klasse behoort en het totaal aantal tuples dat aan de regel voldoet.

Met deze twee onderdelen kunnen we de zoekmethode voeren om zo door de totale zoekruimte op zoek te gaan naar de optimale classificatieregel(s). In Data Surveyor is een zogenaamde beamsearch geïmplementeerd. Dit betekent dat meerdere hill-climbers tegelijkertijd in de zoekruimte op zoek zijn naar het optimum. Op deze manier is er een grote kans dat het echte optimum ook daadwerkelijk gevonden wordt.

Het zal duidelijk zijn dat er nog steeds een enorme boom uit dit algoritme te voorschijn kan komen. Daarom wordt de gebruiker de mogelijkheid geboden de boom in te perken door zowel de breedte als de diepte van de boom op te geven. De breedte geeft aan hoeveel van de interessante buren worden geselecteerd voor verdere analyse. De diepte geeft aan uit hoeveel selectiecriteria een regel maximaal mag bestaan.

In ***Fout! Onbekende schakeloptie-instructie.*** is een boom weergegeven die met behulp van Data Surveyor is gegenereerd. Bij deze boom is een breedte van vijf en een diepte van drie opgegeven. Het doel was in de gegevensverzameling van een verzekeringsmaatschappij op zoek te gaan naar een beschrijving van polisnemers die een grotere kans hebben een ongeluk te veroorzaken. In de knooppunten is het selectiecriterium weergegeven. Onder het selectiecriterium staan de dekking van deze classificatieregel (het pad door de boom naar deze



knoop) in de hele gegevensverzameling en de kwaliteitswaarde (het percentage van deze dekking dat behoort tot de klasse). De regel "`gender = m ∧ age ≤ 22`" heeft bijvoorbeeld een dekking van 1349 tuples in de database en 60,8% van deze dekking behoort tot de doelklasse.

In de volgende paragraaf wordt vooral ingegaan op de werkbelasting die Data Surveyor voor het database management systeem genereert

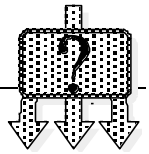
4.4 De rol van het database management systeem

Het database management systeem dient te voorzien in de basisgegevens voor de burengenerator en de kwaliteitscalculator. Deze basisgegevens bestaan allereerst uit de dekking die een regel heeft in de database. Dit is niets anders dan het uitvoeren van de conjunctie van de selectiecriteria die zich in de knooppunten van de regel bevinden.

Om nieuwe buren te genereren dient de generator te weten welke selectiecriteria er in aanmerking komen. Daarnaast dient de kwaliteitsfunctie te weten hoeveel tuples voldoen aan deze selectiecriteria. In deze informatiebehoefte wordt voorzien door het berekenen van frequentieverdelingen. Deze frequentieverdeling geven de verdeling van de waarden van een attribuut in de dekking van een regel weer. De waarden dienen als invoer voor de burengenerator en de frequenties voor de kwaliteitscalculator. De zoekfunctie gaat dan eigenlijk op zoek naar de pieken in deze verdelingen en levert de bijbehorende waarde als selectiecriterium op.

Het zal duidelijk zijn dat de belasting van het database management systeem bij de start van het algoritme het grootst is. Per iteratie neemt de belasting echter af, dit komt door zowel het horizontale als het verticale zoom karakter van het algoritme. Het horizontale zoom gedrag wordt veroorzaakt door de conjunctie van selectiecriteria, we hebben te maken met een deelverzameling van een deelverzameling enzovoorts. Kortom het aantal tuples dat een rol speelt neem steeds verder af. Dit effect is echter wel afhankelijk van de selectiviteit van de gevonden regels. Het verticale zoom effect is het gevolg van het feit dat we steeds minder frequentieverdelingen hoeven te berekenen, aangezien steeds minder attributen nog in aanmerking komen om als een selectiecriterium in een regel voor te komen (omdat ze al in die regel voorkomen).

De specifieke invulling van het berekenen van de frequentieverdelingen wordt behandeld in hoofdstuk 6. In dit hoofdstuk wordt het database management systeem, Monet, dat als backend fungeert voor Data Surveyor beschreven. Hiervoor zal echter in hoofdstuk 5 de theorie van parallelle database management systemen worden besproken.



5. Parallelle Database Management Systemen

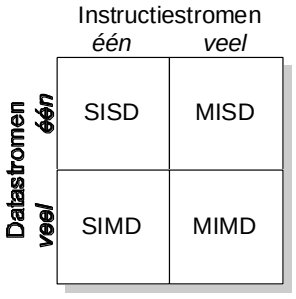
5.1 Inleiding

Zoals in het hoofdstuk over data mining is besproken worden de basis gegevens verzameld in een data warehouse en op deze verzameling gegevens wordt met behulp van slimme leer algoritmes nieuwe informatie uit de opgeslagen feiten afgeleid. Deze algoritmes vuren allerlei vragen op de database af om de volgende stap in hun algoritme te kunnen bepalen. Bij Data Surveyor, besproken in het voorgaande hoofdstuk, zijn dit bijvoorbeeld de frequentie verdelingen om de kwaliteit van kandidaat selectiecriteria te kunnen berekenen. Zo'n vraag wordt afgehandeld door een database management systeem: een verzameling programma's dat de gebruiker in staat stelt een database te creëren, te onderhouden en, niet bepaald de onbelangrijkste, te ondervragen. We hebben gezien dat de gegevens in deze database uit alle hoeken en gaten van de organisatie en daarbuiten afkomstig zijn. Hierdoor is de verzameling omvangrijk en moeten we speciale maatregelen nemen om een redelijke responsietijd te verkrijgen. Een mogelijkheid is meerdere processors tegelijkertijd aan het beantwoorden van de vraag te laten werken, dit is de aanpak die bij parallelle database management systemen wordt gekozen. In dit hoofdstuk zal de theorie achter deze database management systemen, voor zover die van belang is voor deze scriptie, worden behandeld. Voordat we echter software kunnen bouwen die iets parallel door verschillende processors kan laten verwerken dient er parallelle hardware aanwezig te zijn. Daarom begint dit hoofdstuk met een inleiding over parallelle hardware.

5.2 Parallelle hardware

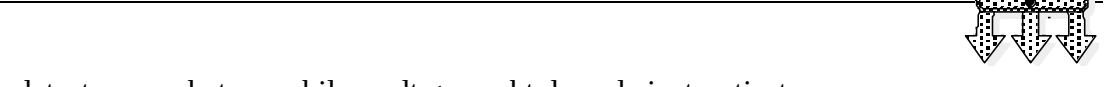
Er zijn verschillende architecturen mogelijk om een aantal processors samen aan één taak te laten werken. De bekendste indeling of taxonomie van deze architecturen is die van Flynn [Flynn, 1966]. Flynn's taxonomie is gebaseerd op twee concepten: instructiestromen en datastromen. Één instructiestroom komt overeen met een serie instructies behorende bij één programmateller. De datastroom bestaat uit de operanden voor deze instructies. Als we deze twee concepten ieder op de as van een quadrant plaatsen dan krijgen we de volgende indeling (zie ook ***Fout! Onbekende schakeloptie-instructie.***):

- Single Instruction Single Data (SISD): een sequentiële computer;
- Single Instruction Multiple Data (SIMD): meerdere processors voeren dezelfde bewerking uit op verschillende gegevens;
- Multiple Instruction Single Data (MISD): in dit geval zouden meerdere processors verschillende bewerkingen uitvoeren op hetzelfde gegeven, deze architectuur wordt over het algemeen als onbruikbaar bestempeld;
- Multiple Instruction Multiple Data (MIMD): autonome processors voeren verschillende bewerkingen uit op verschillende gegevens.



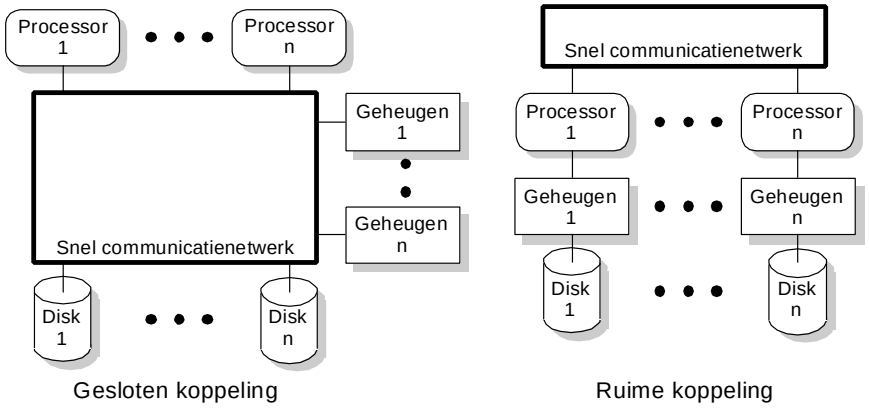
Figuur Fout! Onbekende schakeloptie-instructie. Flynn's taxonomie

Van deze indeling blijken er uiteindelijk maar twee parallel en bruikbaar te zijn, namelijk MIMD en SIMD. Beide werken met



meerdere datastromen, het verschil wordt gemaakt door de instructiestroom.

Haaks op deze indeling staat de indeling naar mate van synchronisatie oftewel hoe vaak worden de processors onderling op elkaar afgestemd. Synchronisatie vindt plaats doordat een gemeenschappelijk besturingsorgaan de processors de opdracht geeft door te gaan met de volgende opdracht. Dit kan voor alle processors dezelfde opdracht zijn, zoals in het geval van een SIMD architectuur, maar ook een andere opdracht zoals in het geval van pijplijn vectorprocessors. In dat geval bestaat er een soort lopende band waarbij elke processor een gespecialiseerde opdracht uitvoert op een gegeven (een vectorelement). Op het moment van synchronisatie schuift het gegeven door naar de volgende bewerkingsstap. Het gaat hierbij om verschillende data en opdrachten per processor, dus dit is een voorbeeld van een MIMD architectuur.

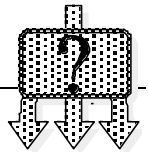


Figuur Fout! Onbekende schakeloptie-instructie. Mate van koppeling

Er zijn meerdere manieren waarop deze synchronisatie tussen twee of meer autonome processors kan plaatsvinden. Men kan boodschappen uitwisselen via een netwerk of een boodschap plaatsen in een gemeenschappelijk gedeeld geheugen. Deze methoden worden weergegeven door de mate van koppeling tussen de processors. In geval van een gesloten koppeling delen de processors gezamenlijk één fysiek geheugen en kunnen via dit geheugen gegevens met elkaar uitwisselen. In het andere geval is er sprake van ruime koppeling. In dit geval heeft elke processor de beschikking over zijn eigen geheugen. Beide vormen zijn weergegeven in ***Fout! Onbekende schakeloptie-instructie.***.

Deze indeling is voldoende voor de hardware die in deze scriptie aan de orde komt. In de loop van de tijd zijn er veel meer architecturen beschikbaar gekomen, sommige van deze architecturen vallen duidelijk binnen één van de groepen van Flynn's taxonomie andere combineren eigenschappen. Om al deze architecturen in een taxonomie te vatten zijn er uitbreidingen op Flynn's taxonomie geformuleerd (zie [Tanenbaum, 1990], [de Waard, 1993] en [Duncan, 1990]).

Hiermee hebben we een parallelle hardware architectuur tot onze beschikking en kunnen we verder gaan met het ontwikkelen van parallelle software. Aangezien we uiteindelijk in



parallelle database management systemen zijn geïnteresseerd wordt de theorie achter deze software in de volgende paragraaf besproken.

5.3 Parallelle database management systemen

Parallellisme kan op de volgende manieren binnen database management systemen bij het verwerken van een vraag worden toegepast:

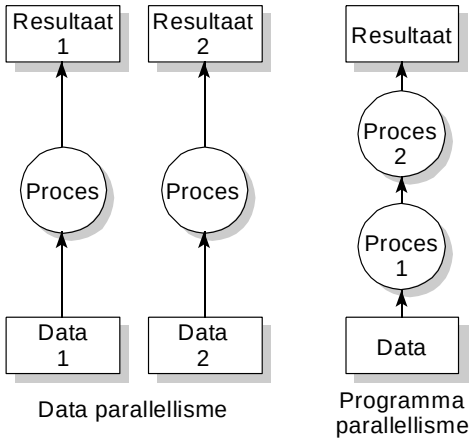
- interquery parallellisme: er worden meerdere vragen tegelijkertijd door het systeem verwerkt;
- intraquery parallellisme: er wordt door meerdere processors tegelijkertijd aan één vraag gewerkt.

Interquery parallellisme wordt al lange tijd door database management systemen aangeboden en dit kan net zo goed op één (via interleaving) als meerdere processors plaats vinden. Voor het versnellen van responsietijden zijn we meer geïnteresseerd in de tweede vorm, intraquery parallellisme.

Binnen intraquery parallellisme kunnen we opnieuw een indeling maken:

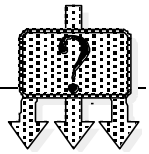
- inter-operatie parallellisme: verschillende relationele operaties binnen een vraag worden tegelijkertijd uitgevoerd;
- intra-operatie parallellisme: één relationele operatie binnen de vraag wordt als meerdere operaties uitgevoerd.

Deze tweedeling komt globaal overeen met de indeling die we ook bij parallelle hardware hebben gemaakt. Inter-operatie parallellisme komt overeen met MIMD, aangezien hier de operaties per processor verschillen. Intra-operatie parallellisme komt dan overeen met SIMD, aangezien de operaties per processor gelijk zijn. De granulariteit van het parallellisme bij MIMD en SIMD is echter veel kleiner (groot: vragen, middel: operaties en klein: instructies). Andere termen voor deze vormen van parallellisme binnen de wereld van database management systemen zijn: programma (inter-operatie) parallellisme en data (intra-operatie) parallellisme. In ***Fout! Onbekende schakeloptie-instructie.*** zijn ze beide afgebeeld.



Figuur Fout! Onbekende schakeloptie-instructie. Vormen van parallellisme

Programma parallellisme maakt vaak gebruik van het pijplijn mechanisme. In een pijplijn zijn er steeds twee processen die een producent en consument paar vormen. De producent bewerkt zelf één voor één de tuples en geeft die door aan de consument die dan de volgende bewerking uitvoert en eventueel als producent voor een andere consument dient. Er ontstaat als het ware een lopende band waarop elke tuple een reeds bewerkingen ondergaat. Deze vorm noemt men een asynchrone pijplijn. Als een producent eerst een hele tabel oplevert voordat de consument aan de slag kan, dan is er sprake van een synchrone pijplijn. Sommige operaties dienen eerst een hele tabel tot hun beschikking te hebben voordat zij zelf tuples kunnen opleveren, denk

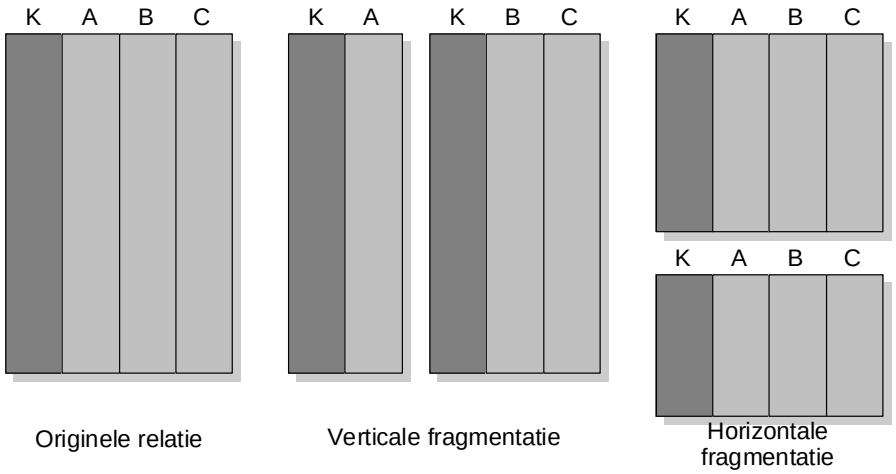


bijvoorbeeld aan een sorteer operatie of een aggregaat. Er ontstaat dus een boomstructuur van synchrone en asynchrone pijplijnen. Het goed managen van deze pijplijnen en de buffering van tussenresultaten is een belangrijk onderwerp bij database management systemen die gebruik maken van programma parallellisme. Voor een uitgebreide beschrijving van het toepassen van programma parallellisme zie [Wilschut, 1993].

In de rest van deze scriptie speelt data parallellisme een belangrijke rol, dit is dan ook de vorm die in de volgende paragraaf uitgebreid wordt besproken.

5.3.1 Data parallellisme

Voordat we data parallellisme kunnen toepassen dienen we ons af te vragen of de relationele algebra dit wel toestaat. Oftewel zijn we in staat met de relationele operatoren dezelfde antwoorden uit een database management systeem gebaseerd op data parallellisme te halen als uit een normaal database management systeem. Hiervoor is het allereerst belangrijk te weten hoe de data verdeeld wordt. Er zijn hierbij in principe twee mogelijkheden: horizontaal en verticaal (zie ***Fout! Onbekende schakeloptie-instructie.*** en [Özsu e.a., 1991]).

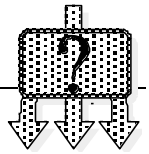


Figuur Fout! Onbekende schakeloptie-instructie. Manieren van fragmentatie

Bij beide methoden geldt dat het mogelijk moet blijven uit de fragmenten de oorspronkelijke relatie te herstellen. Deze operatie waarborgt dan ook dat de relationele operatoren dezelfde antwoorden blijven opleveren. Natuurlijk introduceren deze extra operatoren die door de fragmentatie in de vraag terecht komen, ook nieuwe mogelijkheden om de volgorde van uitvoering te optimaliseren. Deze mogelijke optimalisaties zullen uitgebreid worden besproken in hoofdstuk 9.

De eerste methode is verticale fragmentatie. Hierbij worden de attributen van een relatie verdeeld. De actie die moet worden uitgevoerd om de originele relatie te herstellen is een join. Dit betekent echter wel dat bij alle fragmenten minimaal de primaire sleutel moet worden opgenomen. De reconstructie regel voor verticale fragmentatie ziet er dus als volgt uit:

$R = R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n$



De tweede methode is horizontale fragmentatie. Hierbij worden de tupels verdeeld over de verschillende processors. Dit kan op drie verschillende manieren gebeuren:

- round robin: de tuples worden in een round robin manier over de nodes verdeeld (een voorbeeld bij twee processors: processor één krijgt tuple één, processor twee tuple twee, processor één tuple drie, enzovoort);
- hashing: een hash-functie bepaalt op basis van een attribuut (het fragmentatie attribuut) de positie van de tuple;
- interval: de tuples worden verdeeld aan de hand van verschillende intervallen, elk interval hoort bij een ander fragment en het attribuut dat hier als basis voor dient is het fragmentatie attribuut.

Om de oorspronkelijke relatie te herstellen hoeven de fragmenten alleen met elkaar verenigd te worden. De reconstructie regel ziet er voor horizontale fragmentatie als volgt uit:

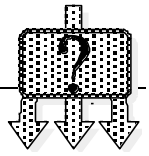
$R = R_1 \cup R_2 \cup R_3 \cup \ldots \cup R_n$

Natuurlijk kunnen deze twee methoden van fragmentatie, de verticale en de horizontale, ook tegelijkertijd worden toegepast.

De volgende vraag is natuurlijk hoe weet ik nou welke fragmentatie methode ik moet hanteren? Bij gedistribueerde databases gebeurt dat op basis van het toegangsgedrag op de tuples in de database. Hierdoor blijven gegevens die zeer plaatselijk van aard zijn, bijvoorbeeld de verkoopgegevens van de lokale vestiging, dichtbij en snel toegankelijk. Bij parallelle databases spelen dit soort geografische overwegingen geen rol meer, de data bevindt zich uiteindelijk toch in “één” machine. De vertragende factor, namelijk het uitwisselen van data tussen servers, blijft echter toch nog een rol spelen. Bij parallelle systemen met een ruime koppeling is de uitwisseling van de data onwenselijk want ze moeten dan over de relatief langzame communicatie lijnen vervoerd worden.

We kunnen dit illustreren aan de hand van het volgende voorbeeld: neem aan dat een systeem processors bevat die 30 operaties per micro seconde uitvoert en het opstarten van een uitwisselingsoperatie (message passing) tussen twee processors in dit systeem kost 3 milli seconde (de network latency). Dit betekent dat voor het opstarten van het verzenden van één boodschap de processor 90.000 operaties had kunnen uitvoeren. Het verzenden van de boodschap moet dus de complexiteit van het probleem verminderen met 90.000 operaties [Sloot e.a., 1995]. Dit illustreert duidelijk dat communicatie tussen de parallelle processors bij ruime koppeling zoveel mogelijk voorkomen moet worden.

Het transporteren van fragmenten wordt nodig als de hele relatie voor een operatie nodig is, zoals het berekenen van een aggregaat. Door het toegangsgedrag tot de fragmenten in een specifieke situatie te bestuderen kunnen de fragmenten zo optimaal mogelijk bij een processor worden geplaatst. In hoofdstuk 9 zal dit voor de onderzoekssituatie worden gedaan. Een ander onderwerp is het aantal tuples dat overgezonden dient te worden zo klein mogelijk te maken. Dit heeft betrekking op de optimalisatie van de uitvoering van algebraïsche operaties, zie paragraaf 2.4. Hoofdstuk 9 richt zich erop de volgorde van algebraïsche operaties, na het



toepassen van de reconstructie regel, te optimaliseren. Hoofdstuk 8 richt zich op manieren om dit bij het berekenen van aggregaten te bereiken.

Nu de data echter verdeeld is blijft nog het afhandelen van de vragen, die aan het database management systeem gesteld worden, over. Hiervoor kunnen we drie verschillende verwerkingsstrategieën onderscheiden [Khoshafian e.a., 1988] (zie ***Fout! Onbekende schakeloptie-instructie.***):

- Remote Access (RA): één processor heeft de uitvoering van de vraag in handen en vraagt aan andere processors om de fragmenten of tuples van de fragmenten als hij ze nodig heeft;
- Execution at Repositories (ER): een vraag wordt opgedeeld in deelvragen die naar alle potentiële processors met een fragment of tuple worden gezonden;
- Dynamic Loading (DL): een vraag wordt opgedeeld in deelvragen die echter pas naar een processor worden gestuurd als bekend is dat zich daar een relevant fragment of tuple bevindt.

De eerste strategie (RA) leidt makkelijk tot een meer sequentiële uitvoering dan het bedoelde parallellisme. De tweede strategie (ER) biedt meer mogelijkheden tot parallellisme, maar heeft tot gevolg dat sommige processors wel aan het werk worden gezet maar uiteindelijk niets zullen opleveren. De derde strategie (DL) heeft dezelfde mogelijkheden tot parallellisme maar bepaald van tevoren of een operatie wel een resultaat zal opleveren, dit voorspellen kan gebeuren door gebruik te maken van statistische gegevens over de fragmenten.

- de vraag luidt σ (R) ✹ S
- R is gefragmenteerd over processor 1 en 2
- S is gefragmenteerd over processor 2 en 3

Strategie 1: Remote Access

	Processor 1	Processor 2	Processor 3
1.	$T_1 = \sigma(R_1)$	$T_2 = \sigma(R_2)$	
2.	ontvang T_2	verzend T_2	
3.	$T = T_1 \cup T_2$		
4.	ontvang S_2	verzend S_2	
5.	ontvang S_3		verzend S_3
6.	$S = S_2 \cup S_3$		
7.	T ✹ S		

Strategie 2: Execution at Repositories

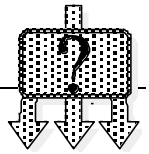
	Processor 1	Processor 2	Processor 3
1.	$T_1 = \sigma(R_1)$	$T_2 = \sigma(R_2)$	
2.	verzend T_1	ontvang T_1	ontvang T_1
3.		verzend T_2	ontvang T_2
4.		$T = T_1 \cup T_2$	
5.		T ✹ S_2	T ✹ S_3

Strategie 3: Dynamic Loading

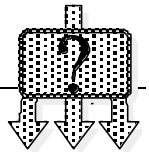
	Processor 1	Processor 2	Processor 3
1.	$T_1 = \sigma(R_1)$	$T_2 = \sigma(R_2)$	
2.	verzend T_1		ontvang T_1
3.		verzend T_2	ontvang T_2
4.			$T = T_1 \cup T_2$
5.			T ✹ S_3

Figuur Fout! Onbekende schakeloptie-instructie. Voorbeeld verwerkingsstrategieën

Deze vorm van parallellisme zal uitgebreider in hoofdstuk 9 aan de orde komen aangezien het onderzoek van deze scriptie zich richt op data parallellisme en daarbinnen op het genereren van de deelvragen. Voor het ontwikkelen van het decompositie algoritme zullen de verschillende verwerkingsstrategieën de revue passeren. In het volgende hoofdstuk worden echter eerst de



specifieke eigenschappen van Monet, het in de praktijksituatie toegepaste database management systeem, beschreven.



6. Monet

6.1 Inleiding

Monet is een uitbreidbaar parallel main memory database management systeem dat dient als backend voor Data Surveyor. In dit hoofdstuk zullen allereerst de eigenschappen van Monet beschreven worden. Aan het einde wordt er aangegeven hoe deze eigenschappen worden ingezet om Data Surveyor te ondersteunen. De belangrijkste eigenschappen van Monet zijn:

- een binair datamodel;
- de hotset bevindt zich in het primaire geheugen;
- inter-operatie parallellisme wordt ondersteund;
- er is de mogelijkheid eigen datatypes en operaties toe te voegen.

In de eerste paragrafen van dit hoofdstuk zullen deze eigenschappen kort belicht worden, waarna deze ook in de architectuur van Monet kunnen worden aangeduid..

6.2 Binair datamodel

Het binaire datamodel of het Decomposed Storage Model (DSM) is eigenlijk een vorm van totale verticale fragmentatie. Dit betekent dat een bestaande relatie wordt gefragmenteerd in een serie fragmenten, Binary Association Tables (BATs) genaamd. Deze BATs bevatten in ieder geval elk één attribuut van de originele relatie. Een BAT bestaat uit een serie Binary UNits (BUNs), overeenkomend met de tuples uit een relationele tabel. Elke BUN bestaat uit een head en een tail. In de head bevindt zich een unieke tuple of object identifier en in de tail een attribuut van de oorspronkelijke relatie. Via de unieke tuple identifier kan worden voldaan aan de eis dat de oorspronkelijke relatie kan worden hersteld, in het geval van verticale fragmentatie door een `semijoin` operatie. Om dit alles te illustreren kunnen we een tabel `Test` nemen, bestaande uit de attributen `naam`, `geslacht` en `leeftijd`. Binaire fragmentatie van de tabel leidt tot de volgende BATs: `Test_naam`, `Test_geslacht` en `Test_leeftijd` (zie ***Fout! Onbekende schakeloptie-instructie.***).

Test

Naam	Geslacht	Leeftijd
Bert	m	25
Ernie	m	6
Sien	v	34
Inimini	v	12

Test_naam

tid	Naam
1	Bert
2	Ernie
3	Sien
4	Inimini

Test_geslacht

tid	Geslacht
1	m
2	m
3	v
4	v

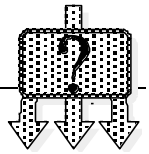
Test_leeftijd

tid	Leeftijd
1	25
2	6
3	34
4	12

Figuur Fout! Onbekende schakeloptie-instructie. Translatie relationele tabel naar binaire tabellen

Wat is het voordeel van deze hele transformatie? Allereerst wordt de implementatie van de relationele operatoren er een stuk simpeler en daarmee ook sneller op. Verder worden alleen die attributen geladen die benodigd zijn, waardoor de hele hotset (de verzameling actieve tabellen) van de database sneller in het primaire geheugen van de computer past.

Binnen database management systemen kunnen de volgende vraagbelastingen worden onderscheiden (zie [Berg, van den, 1994]):



- klasse A: vragen die een groot aantal attributen van een klein aantal tuples manipuleren;
- klasse B: vragen die een klein aantal attributen van een groot aantal tuples manipuleren;
- klasse C: vragen waarbij beide aspecten gemiddeld zijn.

Het binaire datamodel blijkt goed te scoren bij klasse B queries. Dat is logisch aangezien het aantal benodigde `semijoin` operaties toeneemt met het aantal attributen dat een rol speelt bij een vraag. Ook bij het binaire datamodel blijft de `join`/`semijoin` een dure operatie.

6.3 BAT algebra

Op deze BATs biedt Monet een hele serie standaard relationele operaties zoals selectie en vereniging. Alleen de projectie operatie wordt niet aangeboden, aangezien er voor de beantwoording van een vraag alleen gebruik wordt gemaakt van BATs met attributen die van belang zijn. In ***Fout! Onbekende schakeloptie-instructie.*** zijn de operaties die van belang zijn voor deze scriptie weergegeven.

Relationele operatie	BAT operatie	resultaat
$\sigma_{b=T}AB$	<AB>.select(T)	$\{ ab \mid ab \in AB \wedge b = T \}$
$\sigma_{b\geq Tl \wedge b\leq Th}AB$	<AB>.select(Tl,Th)	$\{ ab \mid ab \in AB \wedge b \geq Tl \wedge b \leq Th \}$
$AB \blacksquare_{b=c} CD$	<AB>.join(CD)	$\{ ad \mid ab \in AB \wedge cd \in CD \wedge b = c \}$
$AB \ltimes_{a=c} CD$	<AB>.semijoin(CD)	$\{ ab \mid ab \in AB, \exists\, cd \in CD \wedge a = c \}$
fAB	<AB>.histogram	$\{ bf \mid a \in A, \exists!\ f$ frequentie van b $\}$
$AB \cup CD$	<AB>.union(CD)	$\{ xy \mid xy \in AB \vee xy \in CD \}$

Figuur Fout! Onbekende schakeloptie-instructie. Belangrijke BAT operaties

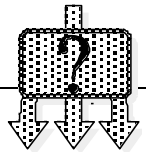
De `semijoin` operatie is een speciale versie van de `join` operatie die vooral geschikt is voor gedistribueerde systemen. Bij een `semijoin` wordt normaliter eerst een projectie gemaakt van de join attributen van de tweede relatie. Deze projectie wordt overgestuurd naar de site van de andere relatie om daar de `join` uit te voeren. Het resultaat omvat dus de tuples van de eerste relatie die voldoen aan de join conditie. In de vorm van relationele operatoren weergegeven ziet dit er als volgt uit:

$R \ltimes_{A=B} S = R \blacksquare_{A=B} (\pi_B S)$

Bij het binaire datamodel is er echter geen sprake meer van een projectie operatie. De implementatie van de `semijoin` operatie komt echter wel overeen met deze definitie, aangezien het resultaat nog steeds de tuples van de eerste relatie die voldoen aan de join conditie omvat.

De gebruiker heeft direct toegang tot de algebraïsche operaties via de Monet Interpreter Language (MIL). Daarnaast kan hij of zij ook toegang tot de database verkrijgen worden via SQL of OQL. De vragen in deze talen worden door utilities vertaald in BAT algebra.

De algebraïsche operatoren worden zonder optimalisatie door de server uitgevoerd. De gebruiker dient dus zelf de logische optimalisatie, zoals selecties voorin de query plaatsen, uit te voeren.



6.4 Main memory DBMS

Zoals hiervoor reeds is aangegeven passen BATs makkelijker in het primaire geheugen van de computer dan een totale relationele tabel met al zijn attributen. Monet's relationele operaties werken alleen op het primaire geheugen en zijn daardoor snel en eenvoudig.

6.5 Inter-operatie parallellisme

Monet is geïmplementeerd op parallelle machines waarbij de processors gesloten gekoppeld zijn, oftewel de processors hebben toegang tot een gedeeld geheugen. Voor deze processors voorziet Monet in de mogelijkheid meerdere algebraïsche operatoren parallel te laten uitvoeren (zie ***Fout! Onbekende schakeloptie-instructie.***). Hierbij wordt geen gebruik gemaakt van een tuple pijplijn, maar de resultaten van een operatie worden helemaal gerealiseerd voordat een volgende operatie ermee aan de gang kan. Dit betekent dat je alleen operaties die niet van elkaar afhankelijk zijn parallel naast elkaar kunt uitvoeren. Wel wordt er in operaties voorzien om een BAT in fragmenten op te splitsen en eenzelfde operatie kan dan op deze fragmenten parallel worden uitgevoerd. Men dient zelf de deelresultaten echter weer samen te voegen. Binnen een parallel blok van een vraag is de vraagsteller ook zelf verantwoordelijk voor het bewaken van de concurrency tussen de processors.

```
{ }  → sequentieel blok
[ ]  → parallel blok
```

```
{
   temp1 := select(Test_geslacht,"m");
   [
      temp2 := semijoin(Test_naam,temp1);
      temp3 := semijoin(Test_leeftijd,temp1);
   ]
   print(temp2,temp3);
}
```

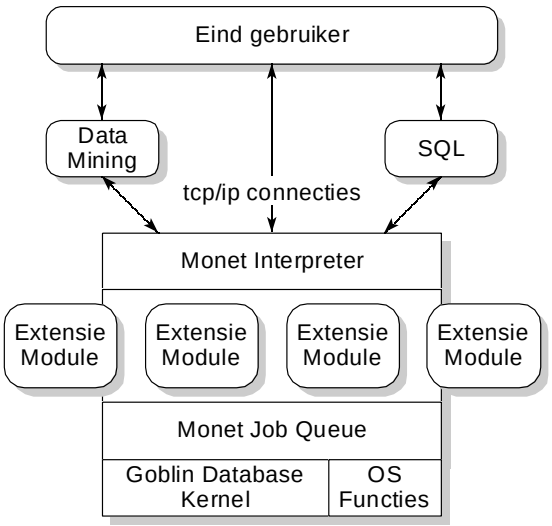
Figuur Fout! Onbekende schakeloptie-instructie. Voorbeeld parallelle vraag

6.6 Uitbreidbaarheid

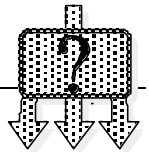
Er wordt voorzien in de mogelijkheid zelf uitbreidingsmodules te specificeren voor Monet. Men kan via zo'n module zelf nieuwe datatypes, bijbehorende accelerators en operatoren toevoegen. Deze modules worden in C of C++ geprogrammeerd en via de Monet Extension Language (MEL) aan Monet gekoppeld.

6.7 De architectuur van Monet

Al de onderdelen van Monet die we hiervoor hebben besproken vinden we terug in de architectuur van Monet (zie ***Fout! Onbekende schakeloptie-instructie.***). De Monet interpreter onderhoudt de verbindingen met verschillende gebruikers en applicaties. De server of de clients kunnen opdracht hebben gegeven extensie modules te laden, deze modules zijn dan via de in MEL gedefinieerde operaties



Figuur Fout! Onbekende schakeloptie-instructie. De architectuur van Monet



beschikbaar in de interpreter. De volgende laag houdt de verschillende jobs bij die uiteindelijk door de Goblin Database Kernel (GDK) worden uitgevoerd. De GDK biedt toegang tot de BATs en BUNs en bevat de eigenlijke implementatie van het binaire model. Monet maakt zoveel mogelijk gebruik van de standaard functionaliteit van het besturingssysteem voor geheugen- en buffermanagement e.d., vandaar dat deze functionaliteit zich ook in deze onderste laag bevindt.

6.8 Monet en Data Surveyor

In deze paragraaf wordt dieper ingegaan op de samenwerking tussen Monet en Data Surveyor. Zoals hiervoor reeds is aangegeven, is Monet zeer geschikt voor het manipuleren van grote gegevensverzamelingen, zolang het aantal attributen dat een rol speelt maar beperkt blijft. Dit lijkt enigszins strijdig met data mining zoals dit eerder is beschreven. In een data warehouse werden namelijk zo veel mogelijk gegevens verzameld over alle entiteiten die een relatie hebben tot een organisatie. Dit leidt tot grote tabellen met zowel veel tuples als veel attributen. Gelukkig hebben we echter te maken met het verticale en horizontale zoom karakter van het data mining algoritme (zie paragraaf 4.4).

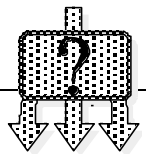
Het horizontale zoom effect, dat er steeds minder tuples een rol spelen, kunnen we uitbuiten door steeds het selectie resultaat van de oorspronkelijke regel, die nu wordt uitgebreid met een nieuw selectie attribuut, te bewaren. Uitbreiding van een regel houdt dus alleen een extra `semijoin` operatie met het nieuwe attribuut in.

Het verticale zoom gedrag houdt in dat er steeds minder attributen in aanmerking komen om bij de regel te worden opgenomen. Als er al een selectie op geslacht plaats vindt heeft het geen nut hierover nogmaals een selectie conditie in de classificatieregel op te nemen. Deze twee effecten hebben tot gevolg dat het aantal attributen dat uiteindelijk een rol speelt toch beperkt blijft.

Monet ondersteunt Data Surveyor doeltreffend op de volgende terreinen:
- het opslaan en toegankelijk maken van de data via BATs;
- het verzamelen van statistische informatie;
- het berekenen van selecties;
- het bewaren van tussenresultaten;
- het voorzien in een parallelle uitvoering.

Het eerste onderdeel spreekt voor zich, voor de andere terreinen is wat meer inzicht nodig in de vragen die de data mining tool genereert. Deze vragen bestaan eigenlijk uit twee fasen, namelijk het bepalen van de dekking van een classificatieregel (de selectie fase) en het berekenen van statistieken over mogelijke uitbreidingen van een regel (de aggregatie fase). Beide fasen worden in één keer in de vorm van een script aan Monet aangeboden.

Als voorbeeld kunnen we kijken naar een gedeelte van de vragen die voor het opbouwen van de boom uit ***Fout! Onbekende schakeloptie-instructie.*** zouden worden gegenereerd. Stel we hebben net gevonden dat de regel "`Age ≤ 21 ∧ Gender = m`" genoeg dekking in de database heeft en in aanmerking komt om uitgebreid te worden. De nog interessante



uitbreidingen voor deze regel moeten gevonden worden in selectiecriteria met de attributen `Town` en `Carprice`. Het eerste wat echter moet gebeuren is de oorspronkelijke regel uit te breiden met de gevonden uitbreiding "`Gender = m`", dit is de selectiefase. De bijbehorende statements zouden er als volgt uitzien:

```
1. tmp0 := select(test_gender,"m");
2. pos_db2 := semijoin(pos_db1,tmp0);
3. neg_db2 := semijoin(neg_db1,tmp0);
4. tmp0.destroy;
```

De BATs `pos_db1` en `neg_db1` bevatten hier de BUNs die voldoen aan de oorspronkelijke classificatieregel (`Age ≤ 21`). Door de `semijoin` blijven alleen de BUNs over die en aan de oorspronkelijke regel voldoen en aan het nieuwe selectiecriterium.

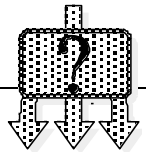
De volgende fase is nu Data Surveyor te voorzien van de data die het nodig heeft om de kwaliteit van een kandidaat uitbreiding uit te rekenen. Dit gebeurt in de aggregatie fase. Hierbij wordt van elk kandidaat attribuut, in dit geval `Town` en `Carprice`, een histogram gemaakt van de verdeling van de waarden van het attribuut in de BUNs die aan de huidige regel voldoen. Voor deze data mining sessie zouden de statements van deze fase er als volgt uitzien:

```
1. tmp1_pos := semijoin(test_town,pos_db2).histogram;
2. tmp1_neg := semijoin(test_town,neg_db2).histogram;
3. tmp2_pos := semijoin(test_carprice,pos_db2).histogram;
4. tmp2_neg := semijoin(test_carprice,neg_db2).histogram;
5. tmp1_pos.print;
6. tmp1_pos.destroy;
7. tmp1_neg.print;
8. tmp1_neg.destroy;
9. tmp2_pos.print;
10.tmp2_pos.destroy;
11.tmp2_neg.print;
12.tmp2_neg.destroy;
```

De `print` statements zorgen voor het uitvoeren van de gegevens van Monet naar Data Surveyor, die de frequentieverdelingen gebruikt om de volgende selectiecriteria te bepalen.

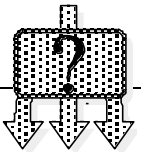
Als de selectiecriteria zijn bepaald wordt in de selectiefase worden de BATs met de tuples die de voldoen aan de verschillende conjuncties van selectiecriteria, uitgebreid met de nieuwe selectiecriteria. De bestanden met de oude conjuncties kunnen verwijderd worden met behulp van de `destroy` operatie.

Als we terugkeren naar de terreinen waarop Monet Data Surveyor ondersteund dan zien we het verzamelen van statistische informatie en het berekenen van selecties duidelijk terug in de twee hiervoor beschreven fasen. Het efficiënt beheren van de tussenresultaten vindt plaats door het opslaan van de subsets van de originele database in tijdelijke bestanden. Deze bestanden worden expliciet verwijderd als ze niet meer nodig zijn en hun namen kunnen hergebruikt worden. Het inter-operatie parallellisme dat Monet ondersteund kan worden



ingezet door de verschillende selecties en de histogrammen parallel te laten berekenen. Al met al voorziet de combinatie van Monet met Data Surveyor in een data mining tool met een goede performance op werkelijke databases.

In het volgende hoofdstuk zal beschreven worden welke wensen er echter nog zijn voor deze samenwerking, waarmee de onderzoeksvraag van deze scriptie zal worden geformuleerd.



7. Onderzoeksvraag: Parallelle Data Mining

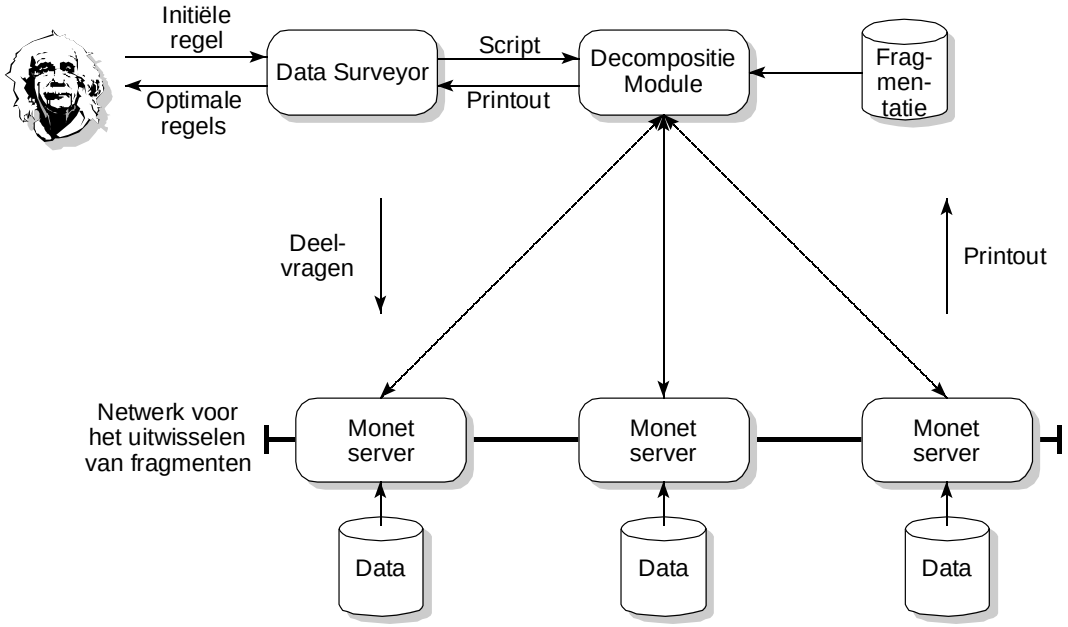
7.1 De onderzoeksvraag

Het vorige hoofdstuk over Monet tezamen met hoofdstuk 4 over Data Surveyor geeft een beschrijving van de huidige praktijksituatie. Er is echter de wens ontstaan om Data Surveyor met Monet als back-end te gaan gebruiken op de SP/2 van IBM. De SP/2 is een parallelle MIMD machine met een ruime koppeling. De parallelle aanpak die op dit moment in Monet is geïmplementeerd ondersteund deze architectuur echter niet (zie paragraaf 6.5).

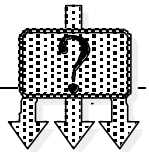
Als we deze gewenste architectuur bekijken dan beschikt iedere processor over zijn eigen primair en secundair geheugen en de processors zijn onderling verbonden door een snel netwerk. We zijn dus in staat op elke processor een eigen Monet database server te laten draaien. Elk van deze servers zal dan beschikken over zijn eigen fragment van de database. Om deze servers samen te laten werken kunnen we gebruik maken van data parallellisme (zie paragraaf 5.3.1).

Door deze opstelling kunnen we het probleem vertalen in de volgende vraagstelling:

> *Hoe verdeel ik een data mining vraag zo optimaal mogelijk over verschillende database servers, gegeven een bepaalde fragmentatie van de originele database, en voer ik deze deelvragen uit zodat er ook daadwerkelijk een reductie van de responstijd wordt verkregen?*



Figuur Fout! Onbekende schakeloptie-instructie. Positie Decompositie Module in de architectuur



Dit is de algemene probleemstelling waarop deze scriptie in de volgende hoofdstukken antwoord probeert te geven. In de praktijksituatie betekent dit dat er een invulling moet worden gezocht voor de decompositie module zoals die in ***Fout! Onbekende schakeloptie-instructie.*** is ingepast in de architectuur van Data Surveyor en Monet.

Een kenmerk van data mining vragen is het berekenen van aggregaten. Aggregaten hebben de eigenschap dat ze de hele relatie in één gegeven, of in ieder geval een klein aantal gegevens, samenvatten. Om zo'n aggregaat te bepalen wordt vaak de originele relatie hersteld uit de verschillende fragmenten en wordt het aggregaat op één processor berekend. Hiervoor dienen alle fragmenten naar één processor overgestuurd worden. Aangezien de originele relatie uit een grote verzameling gegevensverzameling bestaat betekent dit een grote communicatie belasting. Door de berekening van de aggregaten in lokale en globale berekeningen op te splitsen kan deze belasting tot enkele gegevens terug worden gebracht. En vindt de berekening ook parallel plaats. In hoofdstuk 8 zal de theorie achter deze opsplitsing in lokale en globale berekeningen besproken worden.

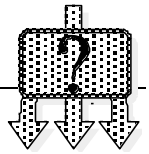
In hoofdstuk 9 wordt dan een methode besproken om de opsplitsing in en uitvoering van de deelvragen zo optimaal mogelijk aan te pakken. Hierbij spelen verschillende onderwerpen een rol. Allereerst dienen de operaties die door de fragmentatie worden toegevoegd zo optimaal mogelijk in de vraag te worden geplaatst. Hierna speelt het probleem welke operaties aan welke processor worden toegewezen. Als de opsplitsing in deelvragen is voltooid speelt nog een optimale uitvoering van deze vragen een rol, hoe voorkom ik zo veel mogelijk wachttijd op de processors. Daarnaast dient er een keuze te worden gemaakt voor een statische of dynamische decompositie methode. In het eerste geval wordt de hele data mining vraag verdeeld op basis van schattingen van de resultaten van de operaties, in het andere geval wordt er gebruik gemaakt van gegevens over de werkelijke resultaten. Zoals zal blijken is de dynamische decompositie methode het beste geschikt voor data mining vragen, aangezien de aannames die ten grondslag liggen aan het bepalen van de schattingen niet gelden voor data mining vragen.

De daaropvolgende hoofdstukken hebben betrekking op de implementatie van hoofdstuk 9 en de testen die met deze implementatie hebben plaats gevonden. Als laatste hoofdstuk treft u dan de conclusies die uit deze resultaten kunnen worden getrokken.

Voordat we echter aan het ontwerpen van een algoritme kunnen beginnen, worden in de volgende paragraaf de randvoorwaarden waaronder dit algoritme dient te functioneren beschreven.

7.2 De randvoorwaarden

De belangrijkste randvoorwaarde heeft betrekking op de fragmentatie van de originele database. Deze fragmentatie dient disjunct en volledig te zijn. Dit betekent dat een tuple uit de database slechts in één fragment mag voorkomen. Daarnaast dienen de fragmenten gezamenlijk de totale originele database te bevatten, oftewel er mogen geen tuples of attributen door de fragmentatie verdwijnen.

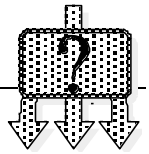


De situatie waarvan wordt uitgegaan is een database die reeds is gefragmenteerd volgens het Decomposed Storage Model. In de vragen die aan het algoritme worden aangeboden bevinden zich reeds de statements (`semijoin` operaties) om deze verticale fragmentatie ongedaan te maken. De horizontale fragmentatiemethode moet worden ondersteund en voor de gebruiker doorzichtig worden gemaakt. Oftewel de gebruiker dient zijn vragen met behulp van de standaard Monet operaties kunnen stellen en zal dezelfde antwoorden ontvangen.

Een volgende randvoorwaarde is dat de database read-only is. Dit betekent dat er geen tuples meer aan toegevoegd of uit verwijderd worden. Onderdelen die we in het algoritme opnemen hoeven dus geen rekening te houden met een tussentijds verandering van de database.

Het algoritme is verder in de eerste plaats bestemd voor data mining vragen. Deze vragen bestaan uit het selecteren van relevante tuples en het berekenen van een aggregaat over deze tuples. Door het Decomposed Storage Model worden deze twee fasen uitgebreid met een derde fase, die tussen de andere twee in komt. In deze fase wordt de selectie uitgebreid tot attributen die van belang zijn voor de aggregatie. Het algoritme dient deze fasen in deze volgorde en de bijbehorende operaties optimaal te ondersteunen.

Met deze uitgangspunten kunnen we een aanvang maken met het ontwikkelen van een antwoord op de onderzoeksvraag.



8. Parallelle Aggregatie

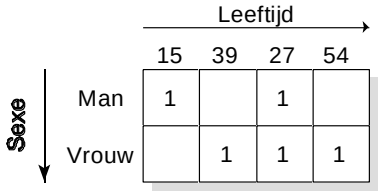
8.1 Inleiding

Aggregaten zijn interessant voor het KDD proces. Zij kunnen gebruikt worden om de eerste twee niveaus van informatie (zie paragraaf 3.7.1) op te diepen en daardoor interessante deelgebieden binnen de data aan te wijzen. Daarnaast kunnen ze ook binnen het data mining proces worden toegepast, bijvoorbeeld in de kwaliteitsfunctie die de waarde van een bepaalde deelverzameling berekent.

Zoals reeds in hoofdstuk 7 is beschreven hebben aggregaten echter de neiging al de fragmenten van de database op één database server bij elkaar te brengen. Aggregaten worden dan als laatste berekend op een vereniging van deze fragmenten. Deze vereniging kan echter uit een flinke hoeveelheid tuples bestaan, zeker als de aggregaat over de hele relatie, in plaats van een selectie van relevante tuples, gaat. Het zou veel efficiënter zijn een deelaggregaat lokaal, en daarmee parallel, te bereken en dit deelresultaat, dat over het algemeen ook veel kleiner, misschien zelfs één tuple, in omvang zal zijn, over te sturen naar de vragende server en daar dan een globaal resultaat te berekenen. Dit betekent echter wel dat we aggregaten dienen op te splitsen in lokale en globale berekeningsstappen. In dit hoofdstuk zal een indeling van aggregaten worden gegeven en voor elk onderscheiden groep in deze indeling zal een opsplitsing worden aangegeven.

8.2 Een indeling van aggregaten

Deze indeling is gebaseerd op een multi-dimensionale data kubus (zie [Gray e.a., 1995]). Dit soort kubussen worden gehanteerd bij het ontsluiten van de gegevens met behulp van OLAP (zie paragraaf 3.7.1). We kunnen het berekenen van aggregaten met behulp van deze kubussen illustreren aan de hand van een twee dimensionaal voorbeeld (zie ***Fout! Onbekende schakeloptie-instructie.***). Voor OLAP is dit eenvoudig te generaliseren naar meerdere dimensies. Voor het parallel berekenen van aggregaten hebben we genoeg aan dit twee dimensionale voorbeeld.



Figuur Fout! Onbekende schakeloptie-instructie. Twee dimensionele gegevensverzameling

Een twee dimensionale data verzameling is als volgt te karakteriseren:

$\{ X_{ij} \mid i = 1,...,I; j = 1,...,J \}$

met:

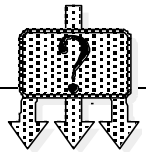
- I: het aantal waarden in eerste dimensie;
- J: het aantal waarden in de tweede dimensie.

Het berekenen van aggregaat F kan nu als volgt worden weergegeven:

$F(\{ X_{ij} \})$

Aggregaat functies die een enkele waarde opleveren worden in de volgende groepen verdeeld:

- distributieve aggregaten;



- algebraïsche aggregaten;
- holistische aggregaten.

Per groep zullen de eigenschappen en de berekening worden toegelicht.

Een aggregaat is distributief als er een functie *G* bestaat waarvoor geldt dat:

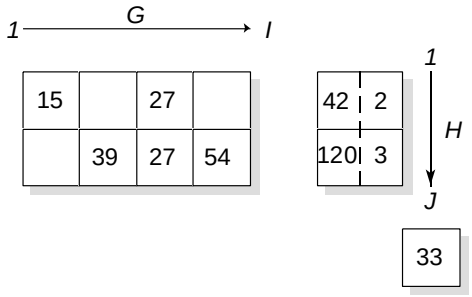
$$F(\{ X_{ij} \}) = G(\{ F(\{ X_{ij} \mid i = 1,...,I \}) \mid j = 1,...,J \})$$

Voorbeelden van distributieve aggregaten zijn de som, de telling, het minimum en het maximum. Voor veel van deze operaties geldt zelfs dat *F* gelijk is aan *G*. Behalve bij de telling, daar is de *G* functie gelijk aan de som.

Algebraïsche aggregaten maken gebruik van twee functies *H* en *G*, voor deze functies geldt het volgende:

$$F(\{ X_{ij} \}) = H(\{ G(\{ X_{ij} \mid i = 1,...,I \}) \mid j = 1,...,J \})$$

Het kenmerk is dat de *G* functie nu meerdere gegevens dient te verzamelen en op te slaan, de *H* functie verwerkt deze gegevens tot het uiteindelijke resultaat. Voorbeelden van algebraïsche aggregaten zijn het gemiddelde (zie ***Fout! Onbekende schakeloptie-instructie.***) en de standaard deviatie. Bij het gemiddelde bepaalt *G* de som en de telling van de deelverzameling. De *H* functie sommeert de gegevens van deze deelverzamelingen en produceert het globale gemiddelde.

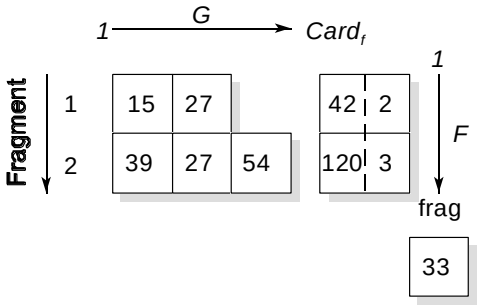


Figuur Fout! Onbekende schakeloptie-instructie. Berekening gemiddelde leeftijd

De holistische aggregaten worden gekenmerkt door het feit dat het onbekend is hoeveel waarden een subaggregaat dient te verzamelen. Het blijkt dat er geen goede opsplitsing voor deze aggregaten te vinden is. Een voorbeeld van deze groep aggregaten is de mediaan.

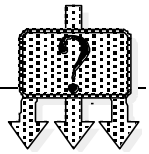
8.3 Opsplitsen in lokale en globale berekeningen

Hoe kunnen we deze indeling nu gebruiken voor de parallelle berekening? Door de fragmentatie als een dimensie te bezien kunnen we de voorgaande indeling hanteren. Als we dan kijken naar de distributieve en algebraïsche aggregaten, dan is daar een duidelijke opsplitsing in lokale en globale berekeningen aanwezig. De binnenste functie is de lokale berekening en wordt parallel uitgevoerd op de fragmenten. De buitenste functie omvat de globale berekening en wordt uitgevoerd met behulp van de resultaten van de lokale berekeningen.



Figuur Fout! Onbekende schakeloptie-instructie. Parallelle berekening gemiddelde leeftijd

We kunnen deze opsplitsing, voor distributieve aggregaten, als volgt weergeven (zie ***Fout! Onbekende schakeloptie-instructie.***):



$F(\{ X_{if} \}) = G(\{ F(\{ X_{if} \mid i = 1,..., card_f \}) \mid f = 1,..., frag \})$
met:

- $card_f$: de cardinaliteit van fragment f;
- $frag$: het aantal fragmenten

Voor de algebraïsche aggregaten ziet deze opsplitsing er als volgt uit:
$F(\{ X_{if} \}) = H(\{ G(\{ X_{if} \mid i = 1,..., card_f \}) \mid f = 1,..., frag \})$

Voor de holistische aggregaten is geen lokale berekening te bepalen. Om deze aggregaten te kunnen bepalen dienen de fragmenten dus op één server verzameld te worden, waar dan de aggregaat berekend kan worden.

Naast aggregaten die een enkele waarde opleveren hebben we nog te maken met de aggregaten die een serie waarden, zoals bijvoorbeeld de frequentieverdeling. Dit levert niet veel wijzigingen op ten opzichte van de behandelde opsplitsing. De lokale operatie levert dan een serie waarden op en de globale operatie aggregeert deze waarden per groep. Als extra parallelle operatie kunnen we een knoop operatie toevoegen die de waarden van de lokale operaties per groep samenvoegt. Bij het gemiddelde leveren de lokale operaties een aantal waarden, namelijk som en telling, op. De knoop operatie sommeert dan de tellingen en sommen per waarde groep. De globale operatie berekent dan per waarde groep het gemiddelde.

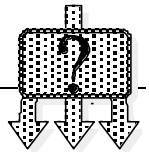
Deze opsplitsing in lokale en globale operatie komt overeen met een algemeen gehanteerde strategie binnen data parallellisme, namelijk de “verdeel en heers” strategie. Voor informatie over deze strategie en afgeleiden daarvan zie [Hill, 1994].

8.4 Parallel sorteren

Wat tot op dit moment niet aan de orde is gekomen is de volgorde van de gegevens. Bepaalde aggregaten, zoals de mediaan of een cumulatieve som, zijn afhankelijk van de volgorde van de data. Deze volgorde zal vaak niet overeen komen met de volgorde van de sleutelwaarden. Voordat het aggregaat kan worden berekend dienen de gegevens dus gesorteerd te worden. Hoe kunnen we deze sortering parallel uitvoeren?

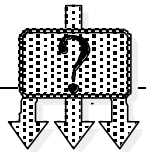
Een eerste mogelijkheid is de “verdeel en heers” strategie toe te passen. Hiervoor dienen we de handelingen op te splitsen in lokale, knoop en, eventueel, globale stappen. In dit geval kunnen we een snel sorteer algoritme, zoals quicksort (zie [Stubbs e.a., 1989]), voor de lokale sortering gebruiken. Voor het samenvoegen van twee gesorteerde fragmenten kan gebruik worden gemaakt van mergesort (zie nogmaals [Stubbs e.a., 1989]). Deze knoop operaties kunnen in de vorm van een bushy-tree (zie paragraaf 9.4) uitgevoerd worden.

Een andere mogelijkheid is de gesorteerde relatie ook gefragmenteerd op de verschillende servers te laten staan. Hiervoor dienen sorteer operaties afgewisseld te worden door uitwisselingsoperaties. Het probleem met deze methode is dat het aantal benodigde sorteerrondes onvoorspelbaar is en de uiteindelijke kosten van het parallel sorteren zou heel



goed veel groter kunnen zijn dan de hiervoor beschreven methode. Het oplossen van dit probleem is een onderzoek op zich waard en is dan ook opgenomen bij de onderwerpen in hoofdstuk 13.

We kunnen deze theorie over het opsplitsen van de berekening van aggregaten in lokale en globale stappen nu toepassen bij het verdelen van de originele data mining vraag in deelvragen voor de verschillende processors. In het volgende hoofdstuk zal een algoritme ontwikkeld worden dat deze opsplitsing en decompositie uitvoert en daarbij de fragmentatie en belasting van de processors in aanmerking neemt.



9. Dynamische Decompositie Module

9.1 Inleiding

In dit hoofdstuk zal evolutionair een algoritme worden ontwikkeld dat in staat is een data mining vraag, op basis van de fragmentatie, in deelvragen over de verschillende processors te verdelen, met als doel het reduceren van de responstijd. Het algoritme zal verschillende beslissingen dienen te nemen, waarvan de belangrijkste het toewijzen van deelvragen aan de verschillende processors zal zijn. Deze allocatie of toewijzing heeft namelijk de grote invloed op de uiteindelijke responstijd. Om hierover beslissingen te nemen beschikt het algoritme over een model. Reeds eerder in deze scriptie hebben we geconcludeerd dat een model altijd een simplificatie van de werkelijkheid is en daarom nooit voor honderd procent de werkelijke wereld kan weerspiegelen. De betrouwbaarheid van dit model heeft dan ook grote invloed op de geldigheid van de beslissingen die het algoritme neemt. In de volgende paragraaf worden de verschillende onderdelen die een rol binnen het model spelen op een rijtje gezet en worden de aannames die eraan ten grondslag liggen gespecificeerd.

In de daarop volgende paragrafen wordt het algoritme stap voor stap ontwikkeld. Hiervoor maken we in eerste instantie gebruik van de drie methoden voor verwerkingsstrategieën van database vragen voor gefragmenteerde databases zoals die reeds in hoofdstuk 5 zijn beschreven. Deze methoden waren:

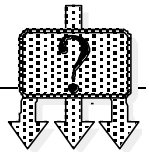
- Remote Access;
- Execution at Repositories;
- Dynamic Loading.

Deze methoden maken in een oplopende mate efficiënt gebruik van parallellisme en communicatie tussen de servers. In feite kunnen we op deze manier evolutionair een algoritme ontwikkelen. Naast deze drie executie strategieën kunnen we echter nog meer onderdelen van de architectuur gebruiken om de parallelle uitvoering van de originele vraag te optimaliseren.

Het hoofdstuk wordt afgesloten met een beschrijving van het complete algoritme zoals dat er uiteindelijk uitziet.

9.2 Het model

Het model van de werkelijkheid dat door het algoritme wordt gebruikt bestaat uit verschillende onderdelen. Een belangrijk onderdeel van het model zijn de schattingsfuncties. Deze schattingsfuncties zijn gebaseerd op een uniforme verdeling van de attribuutwaarden, oftewel elke waarde komt even vaak voor (zie paragraaf 9.4). Dit soort verdelingen komt eigenlijk alleen voor in kunstmatige databases. Een werkelijke database laat zich beter modelleren door een Zipf verdeling (zie paragraaf 9.11). Het probleem met deze verdeling is echter dat de mate van scheefheid bekend dient te zijn en bij welke waarden de pieken en de dalen zich in de database bevinden. Het volgende probleem is dan hoe deze scheefheid wordt geërfd door deelverzamelingen van de originele verzameling. Schattingen op basis van een uniforme verdeling vormen dan ook een gulden middenweg waarbij de schatting gemiddeld even vaak positief als negatief uit kan pakken.



Op basis van de schattingen worden de kosten van de verschillende operaties geschat (zie paragraaf 9.3 en 9.10). Deze kostenfuncties zijn zeer eenvoudig. Aangezien we te maken hebben met een main memory database management systeem kunnen we aannemen dat de gegevens zich in het primaire geheugen bevinden. Hierdoor kunnen we een belangrijk onderdeel van de kosten in een conventioneel database management systeem, namelijk i/o kosten buiten beschouwing laten. Daarnaast heeft de omvang van de gegevensverzameling, het aantal tuples, de grootste invloed op de kosten van de operatie.

Deze kostenfuncties worden uiteindelijk gebruikt bij het toewijzen van de operaties aan bepaalde processors. Deze toewijzing vindt plaats met behulp van allocatieregels (zie paragraaf 9.4, 9.5 en 9.10). Deze allocatieregels modelleren het gedrag van de processors ten opzichte van elkaar, bijvoorbeeld de mate van synchronisatie die er onderling plaatsvindt. Het beeld dat dit totale model oplevert leidt uiteindelijk tot de toewijzing van de operatie aan een specifieke processor, die leidt tot de kleinste stijging in responstijd.

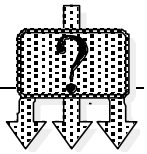
Dit hele model hangt uiteindelijk af van de schattingen van de cardinaliteiten van de verschillende eindresultaten. Zoals hiervoor is beschreven zijn deze schattingen echter uitermate afhankelijk van de specifieke database. Een oplossing voor dit probleem is de werkelijke resultaten van eerdere operaties een belangrijke rol in de allocatie van volgende operaties te laten spelen. Hierdoor wordt het gedrag van het algoritme dynamisch, aangezien de uitkomst door de veranderende werkelijkheid wordt beïnvloed (zie paragraaf 9.11).

In de volgende paragrafen zal de behoefte aan deze verschillende onderdelen van het model en hun rol in het algoritme duidelijk worden. Om de werking van de verschillende versies van het algoritme te verduidelijken zal er een voorbeeldvraag worden gevolgd. In de volgende paragraaf zal deze vraag eerst worden geïntroduceerd.

9.3 Voorbeeld vraag

De voorbeeldvraag dient natuurlijk al de fasen van een data mining vraag, en daarmee ook van de door Data Surveyor gegenereerde vragen, te bevatten. Als basistabel nemen we een tabel met twee attributen, geslacht en leeftijd. Deze tabel is gefragmenteerd over drie processors (zie ***Fout! Onbekende schakeloptie-instructie.***), de doeltreffendheid van deze fragmentatie zal verderop in dit hoofdstuk onder de loep worden genomen. Verder zal het nut van deze fragmentatie gegevens blijken tijdens het ontwikkelen van het algoritme.

De cardinaliteit van de tabellen zal in werkelijkheid vele malen groter (bijvoorbeeld een duizend keer) zijn, maar om de grafische voorstellingen overzichtelijk te houden is er voor deze kleinere cardinaliteit gekozen.



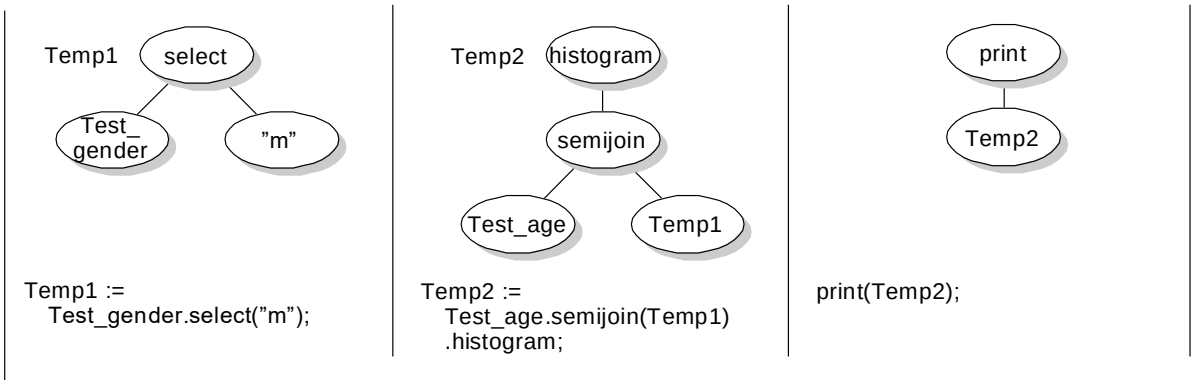
Processor	BAT	Head		Tail		Cardinaliteit	Ordinaliteit
		minimum	maximum	minimum	maximum		
1	Test_gender	1	500	“f”	“m”	500	2
1	Test_age	1	500	12	63	500	12
2	Test_gender	501	1359	“m”	“m”	700	1
2	Test_age	501	1000	23	56	500	25
3	Test_gender	574	1500	“f”	“f”	300	1
3	Test_age	1001	1500	33	78	500	19

Figuur Fout! Onbekende schakeloptie-instructie. Fragmentatie test tabellen

De vraag zelf luidt als volgt:

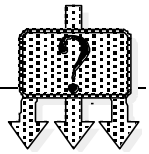
```
1. Temp1 := select(Test_gender,”m”);
2. Temp2 := semijoin(Test_age,Temp1).histogram;
3. print(Temp2);
```

Het eerste statement bevat de selectie fase. In het tweede statement bevindt zich zowel de join fase als de aggregatie fase. Het derde statement zorgt voor de uitvoer van het resultaat naar de gebruiker. Een grafische voorstelling van deze statements kan soms verhelderend werken. Voor deze voorstelling zal gebruik worden gemaakt van de boomstructuren zoals die zijn weergegeven in ***Fout! Onbekende schakeloptie-instructie.***. Omdat deze representatie grafisch omvangrijk wordt als het aantal knooppunten en bladeren toeneemt zullen ze slechts beperkt gebruik worden om de ontwikkeling van het algoritme te illusteren.



Figuur Fout! Onbekende schakeloptie-instructie. Boomrepresentatie van de voorbeeldvraag

Deze boomstructuur, een zogenaamde query tree, wordt in de database wereld gebruikt voor de optimalisatie van vragen, zoals die in paragraaf 2.4 is beschreven. Met behulp van de query tree wordt de volgorde afhankelijkheid van de operaties weergegeven. Optimalisatie vindt nu plaats door op basis van de transformatie regels een query tree te transformeren in een equivalente, maar goedkoper uit te voeren, query tree. Met behulp van de transformatieregels worden de operaties in de knooppunten verplaatst, de volgorde van operaties veranderd dus, de transformatieregels garanderen echter dat het resultaat van de query tree hetzelfde blijft.



Zoals reeds in hoofdstuk 6 is aangegeven voert Monet de operaties uit zonder optimalisatie, de optimalisatie wordt namelijk overgelaten aan de gebruiker zelf. Via het te ontwikkelen algoritme willen wij de gebruiker afschermen van de fragmentatie van de database. Dit heeft tot gevolg dat de gebruiker zelf bepaalde optimalisaties niet kan maken. Zoals in het vervolg van dit hoofdstuk zal blijken richt het algoritme zich erop deze optimalisaties alsnog aan te brengen.

Om het effect van het, tot dan toe ontwikkelde, algoritme zichtbaar te maken wordt er per uitvoeringsstrategie een grafiek afgebeeld. In deze grafiek wordt per processor aangegeven hoe zwaar de belasting is van een bepaalde operatie. Hierdoor wordt via deze grafiek zichtbaar wat de responstijd is van de totale vraag, namelijk de responstijd van de processor met de zwaarste belasting. Omdat bij de werkelijke kosten de kosten van de operaties zover uiteen lopen dat de grafiek overheerst zou worden door één operatie (de `union`) is ervoor gekozen de voorbeeldvraag te illustreren met behulp van de volgende gesimplificeerde kostenfuncties:

```
1.select(n) := n
2.semijoin(n,m) := n + m
3.histogram(n) := n
4.union(n,m) := n + m
5.histosum(n) := n
6.import(n) := n
7.export(n) := n
```

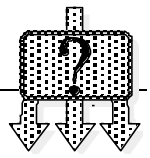
Dit leidt er toe dat op de *x*-as van de grafiek geen geschatte responstijd staat weergegeven, maar de geschatte belasting van de processors in een ongedefinieerde grootheid. Deze simplificatie is alleen aangebracht om de illustratie van de verschillende algoritmes inzichtelijk te houden. In de uiteindelijke implementatie zullen deze kostenfuncties er heel anders uit zien.

In de volgende paragraaf wordt de eerste stap bij het ontwikkelen van het algoritme gemaakt.

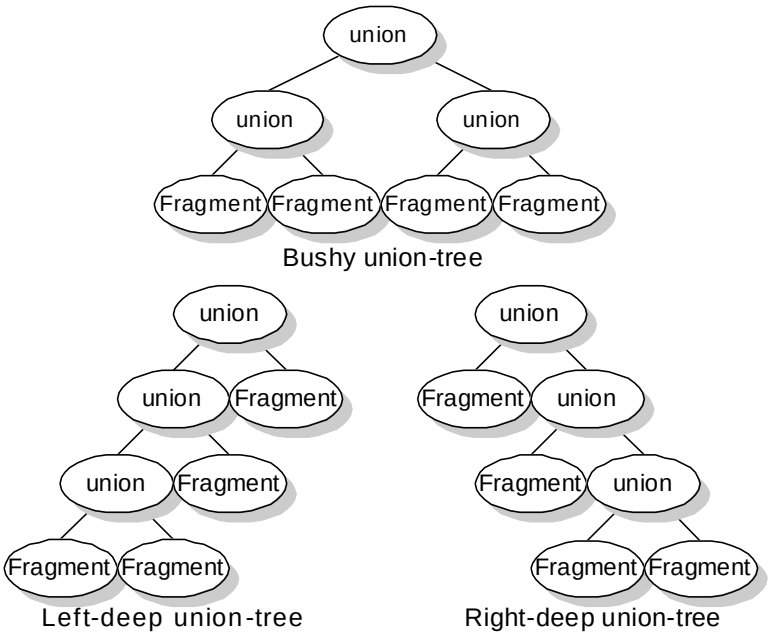
9.4 Remote Access

De eenvoudigste vorm van het uitvoeren van een vraag op een gefragmenteerde database is Remote Access. Bij deze vorm heeft één processor de uitvoering van de vraag in handen en haalt daarvoor alle benodigde fragmenten over van de andere processors. In principe leidt deze strategie tot het vervangen van de verwijzingen naar de originele tabellen door de reconstructie regel om deze tabellen samen te stellen uit de verschillende fragmenten. In dit geval zal het gaan om het toepassen van een serie `union` operaties (zie de beschrijving van horizontale fragmentatie in paragraaf 5.3.1).

De volgende vraag is nu hoe voeren we deze serie `union` operaties op een handige manier uit oftewel waar plaatsen we de haakjes. Er zijn namelijk verschillende manieren om de `union`'s aan elkaar koppelen: een left-deep tree, een right-deep tree of een bushy tree (zie ***Fout! Onbekende schakeloptie-instructie.***).



Aangezien bij Monet de tussenresultaten helemaal gerealiseerd worden voordat de volgende operatie wordt opgestart, biedt de bushy union-tree de meeste mogelijkheden tot parallellisme. Een left-deep of right-deep tree zou tot een sequentiële uitvoering leiden (een parallel database management systeem dat een tuple pijplijn ondersteund zou echter ook een left-deep of right-deep tree parallel kunnen uitvoeren).



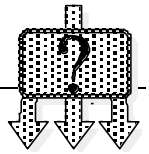
Figuur Fout! Onbekende schakeloptie-instructie. Soorten union-trees

De fragmenten die een rol spelen bevinden zich allemaal op een afzonderlijke processor. De vraag is nu waar wordt de `union` operatie uitgevoerd? Het uitvoeren van alle union operaties op de coördinerende processor berooft ons van alle mogelijkheden van parallellisme. Om de `union` operaties op verschillende processors uit te kunnen voeren hebben we een allocatie regel voor de `union` operatie nodig.

Één van de fragmenten zal voor deze uitvoering verzonden moeten worden. Aangezien het doel is zo min mogelijk communicatie tussen de processors te genereren, zullen we het kleinste fragment oversturen. Deze allocatie regel kan als volgt in pseudo code worden weergegeven:

```
BAT[proc] := allocateUnion(BAT1[proc1,card1],BAT2[proc2,card2]) {
     als:   proc1 == proc2
     dan:   proc := proc1
     anders:als:   card1 ≤ card2
            dan:   proc := proc1
            anders:proc := proc2
}
```

Dit betekent echter wel dat we de cardinaliteit van een fragment dienen te weten. Daarom leggen we een database met statistische gegevens over de fragmenten aan. Deze fragmentatie database zal door het algoritme worden gebruikt voor het alloceren van de originele of toegevoegde statements in de data mining vraag. Deze fragmentatie database is reeds weergegeven in ***Fout! Onbekende schakeloptie-instructie.***.



Om een `union` die als invoer de resultaat BAT van een andere operatie uit de boom krijgt, te kunnen alloceren dient er een schatting te worden gemaakt van het resultaat van een operatie. Hiervoor kunnen we gebruik maken van de in ***Fout! Onbekende schakeloptie-instructie.*** afgebeelde formules.

$$card(\sigma_{b=T} AB) = \frac{card(AB)}{ord(B)}$$

$$card(\sigma_{b\geq Tl \wedge b\leq Th} AB) = \frac{Th - Tl}{\max(B) - \min(B)} \times card(AB)$$

$$card(AB \ltimes_{a=c} CD) = \min(card(AB), card(CD))$$

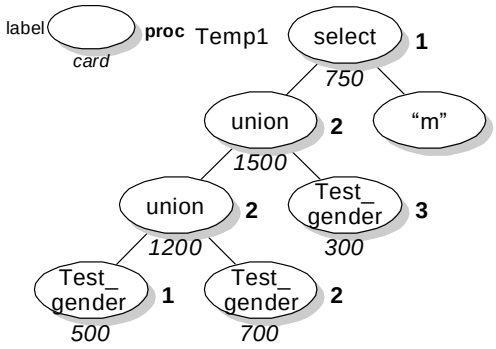
$$card(f\,AB) = ord(B)$$

$$card(AB \cup CD) = card(AB) + card(CD)$$

Figuur Fout! Onbekende schakeloptie-instructie. Schatters voor BAT operaties

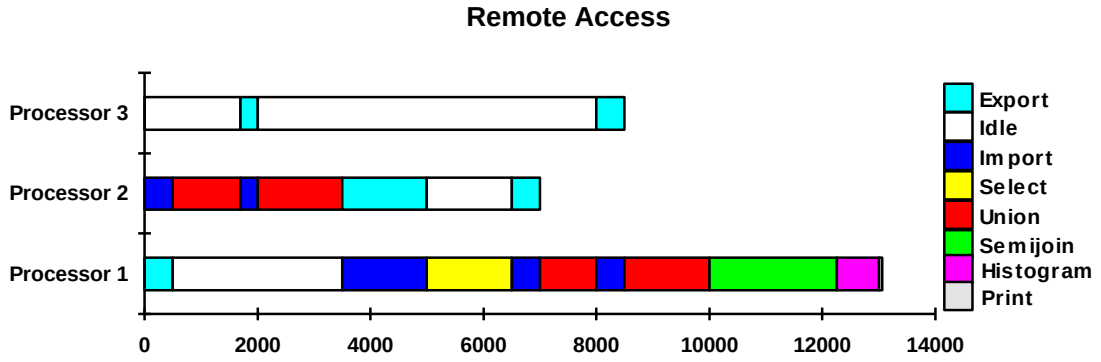
Met behulp van de schattingen van de resultaten en de allocatieregel voor de `union` operaties kunnen knooppunten in de boom aan processors worden toegewezen. Voor het eerste statement uit de voorbeeldvraag leidt dit tot de in ***Fout! Onbekende schakeloptie-instructie.*** afgebeelde boom. Hierbij is processor één als de coördinerende processor aangewezen.

Door de allocatie is ook direct de communicatie tussen de processors duidelijk. Twee met elkaar verbonden knooppunten waarvan de toegewezen processors verschillend zijn wijzen op een uitwisseling van een fragment of een tussenresultaat. In ***Fout! Onbekende schakeloptie-instructie.*** gebeurt dit met het resultaat van de bovenste `union` operatie. Dit resultaat bevindt zich op de tweede processor terwijl de `select` operatie op de eerste processor plaats zal vinden, dit resultaat dient dus van de tweede naar de eerste processor verzonden worden.



Figuur Fout! Onbekende schakeloptie-instructie. RA uitvoering van het select statement

We zijn nu in staat om deze uitvoeringsstrategie te gebruiken om de voorbeeldvraag over meerdere processors te verdelen. De uitvoering van de voorbeeldvraag heeft dan, geschat met behulp van de eerder aangegeven kostenfuncties, het verloop dat is weergegeven in de grafiek in ***Fout! Onbekende schakeloptie-instructie.***.



Figuur Fout! Onbekende schakeloptie-instructie. RA uitvoering van de voorbeeldvraag

Zoals duidelijk te zien is het parallellisme bij deze uitvoeringsstrategie vrijwel nihil. De processors zijn alleen gelijktijdig aan het werk als er fragmenten of tussenresultaten onderling worden uitgewisseld. Bij een hogere fragmentatie graad van de oorspronkelijke database zal dit wel enigszins verbeteren, aangezien er dan meer `union` operaties zijn om te verdelen.

In de volgende paragraaf zullen we Execution at Repositories bespreken die ons weer een stap dichter bij een werkelijk parallelle uitvoering zal brengen.

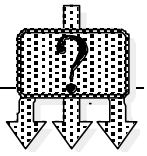
9.5 Execution at Repositories

In de voorgaande paragraaf is er maar zeer beperkt parallel gebruik gemaakt van de processors, daarnaast is de communicatielast van het oversturen van de fragmenten fors. In deze paragraaf willen we daarom de volgorde van de statements verder optimaliseren. Hiervoor zullen we gebruik maken van de transformatie regels zoals die in paragraaf 2.4 zijn geïntroduceerd.

Bij de Remote Access strategie hebben we de verwijzingen naar de originele tabellen, via de reconstructie regel van de fragmentatie methode, vervangen door een union-tree van fragmenten. We kunnen nu gebruik maken van de distributieve eigenschappen van operaties. De `semijoin` operatie is bijvoorbeeld, net als de `join` operatie, distributief over een `union` operatie:

$(AB_1 \cup AB_2) \ltimes CD_3 = (AB_1 \ltimes CD_3) \cup (AB_2 \ltimes CD_3)$

Deze eigenschap kunnen we gebruiken om de `semijoin` operatie als een aantal operaties lager in de query tree onder te brengen, waardoor er een grotere kans ontstaat dat de operatie parallel op verschillende processors wordt uitgevoerd.



Dezelfde methode kunnen we gebruiken bij de `select` operatie. In paragraaf 2.4 is namelijk een heuristische regel gegeven die de selectie operatie distribueert over een `union` operatie:

$$\sigma_q(AB_1 \cup AB_2) \equiv (\sigma_q AB_1) \cup (\sigma_q AB_2)$$

Wat ons nu nog rest is hetzelfde te doen voor de `histogram` operatie. Dit is één van de aggregaten waarvoor we in het voorgaande hoofdstuk een parallelle berekeningsmethode hebben besproken. De `histogram` operatie behoort tot de aggregaten die niet een enkele waarde maar meerdere waarden oplevert. De lokale berekening benodigd voor het berekenen van de frequentieverdeling is de tel operatie per groep. Deze lokale berekening komt overeen met de originele `histogram` operatie. De globale operatie bestaat uit het optellen van de lokaal berekende groepstellingen. Deze globale operatie, de `histosum` operatie (zie de definitie in ***Fout! Onbekende schakeloptie-instructie.***), kan net als de `union` operaties in een bushy-tree georganiseerd worden (zie ***Fout! Onbekende schakeloptie-instructie.***).

Relationele operatie	BAT operatie	resultaat	schatter
$\sum AB$	<AB>.histosum	$\{ a\sum \mid a \in A, \exists! \sum \text{ sommatie van b} \}$	$card(\sum AB) = card(AB)$

Figuur Fout! Onbekende schakeloptie-instructie. Histosum operatie

We kunnen deze verdeling in globale en lokale operaties ook in de vorm van een transformatieregel opschrijven. Deze transformatieregel ziet er dan als volgt uit:

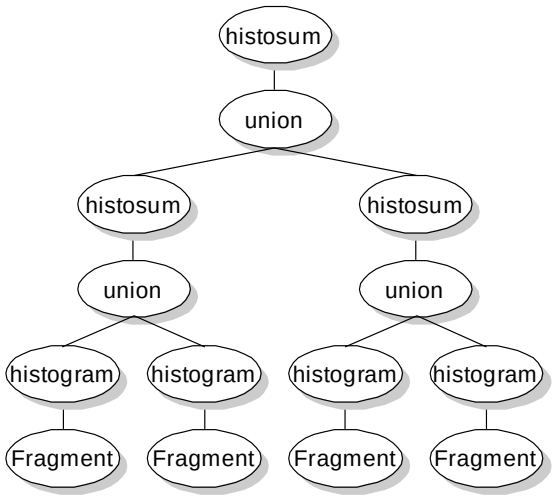
$$f(AB_1 \cup AB_2) \equiv \sum(f(AB_1) \cup f(AB_2))$$

We kunnen de operaties nu verspreiden over meerdere processors. Hiervoor hebben we opnieuw een allocatie regel nodig, in dit geval zelfs twee. De eerste allocatie regel heeft betrekking op de unaire operaties, oftewel operaties, zoals `select` en `histogram`, die slechts op één BAT werken. De allocatie regel voor deze operaties is simpel: voer ze uit op de processor waar zich ook de data bevindt. We kunnen deze regel als volgt in pseude code weergeven:

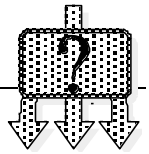
```
   BAT[proc] :=
allocateUnary(BAT1[proc1]) {
        proc := proc1
   }
```

De allocatie regel die we voor de `union` operatie hebben gebruikt kunnen we ook voor de andere binaire operaties hanteren. In pseudo code luidt deze regel als volgt:

```
   BAT[proc] :=
allocateBinary(BAT1[proc1,card1],
BAT2[proc2,card2]) {
        als:    proc1 == proc2
        dan:    proc := proc1
        anders:als:    card1 ≮ card2
```



Figuur Fout! Onbekende schakeloptie-instructie. Query tree voor een parallel berekend histogram



```
                dan:    proc := proc1
                anders:proc := proc2
    }
```

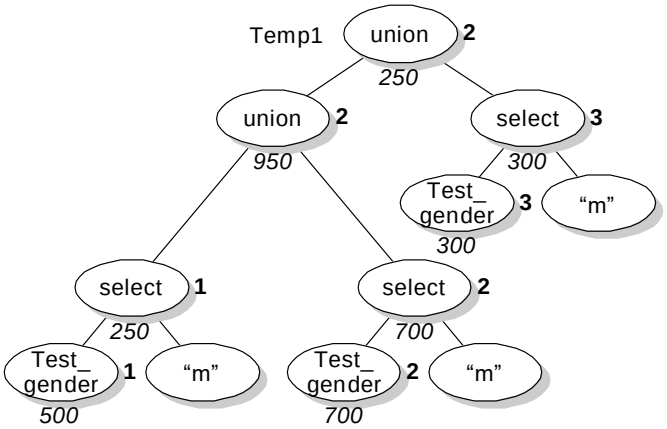
Door de allocatie van binaire operaties kunnen operaties die lager in de query tree terecht gekomen zijn, toch op één processor terecht komen. Over het algemeen zal het goedkoper zijn de operanden voor deze operaties via `union`'s te verenigen en de operatie slechts één keer uit te voeren.

Als we deze transformatie regels en allocatie regels op de voorbeeldvraag toepassen ziet de query tree van de `select` operatie uit de vraag er uit als afgebeeld in ***Fout! Onbekende schakeloptie-instructie.***. De originele `select` operatie is nu over meerdere processors verspreid.

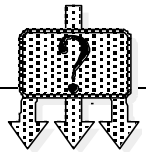
De Execution at Repositories is met behulp van dezelfde kostenfuncties gesimuleerd en levert het resultaat zoals dat is afgebeeld in de grafiek in ***Fout! Onbekende schakeloptie-instructie.***.

Figuur Fout! Onbekende schakeloptie-instructie. ER uitvoering van de voorbeeldvraag

De `select` operatie wordt nu inderdaad parallel uitgevoerd. De `semijoin` operaties zijn wel naar beneden gedrukt in de query tree. Helaas worden ze allen gealloceerd op dezelfde processor, waardoor de operanden via union-trees worden samengevoegd en er één `semijoin` operatie overblijft. Deze allocatie is het gevolg van het grote resultaat (geschatte omvang 750 tuples) van de `select` operatie dat zich op processor twee bevindt, deze hoge cardinaliteit trekt de vervolg operatie, de `semijoin`, naar zich toe. Verderop in dit hoofdstuk zal er een



Figuur Fout! Onbekende schakeloptie-instructie. ER uitvoering van de select operatie



oplossing voor dit probleem worden gevonden. Maar eerst zal de volgende uitvoeringsstrategie, namelijk Dynamic Loading, toegepast worden.

9.6 Dynamic Loading

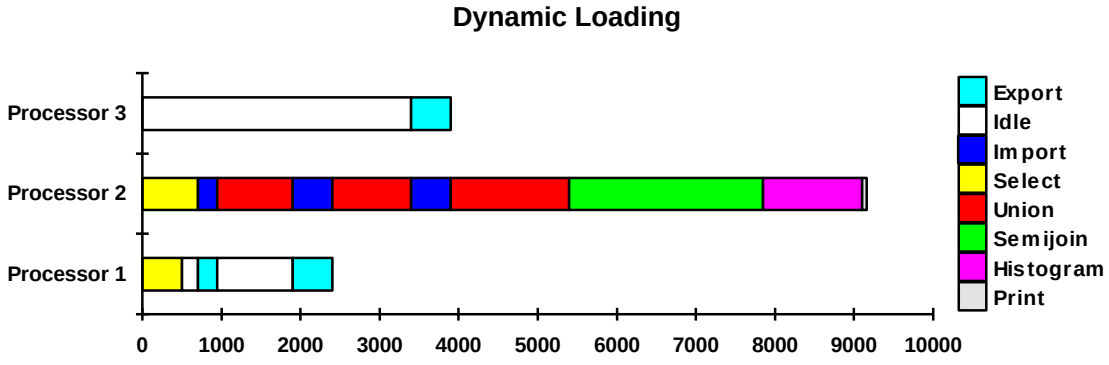
Als we kijken naar de voorbeeldvraag en de fragmentatie dan is duidelijk zichtbaar dat sommige operaties helemaal geen resultaat zullen opleveren. Zo zal de selectie van de mannen op het fragment van `Test_gender` op processor 3 niets opleveren, aangezien dit fragment alleen vrouwen bevat. Door te controleren of het selectie criterium wel een overlap heeft met een bepaald fragment zouden we deze `select` operatie achterwege kunnen laten. Voor deze controle dienen we het interval te weten die door waarden in de tail van het fragment wordt overlapt. Dit interval wordt weergegeven door de minimale en maximale waarde in de tail van het fragment, deze informatie is reeds opgenomen in de fragmentatie database (zie ***Fout! Onbekende schakeloptie-instructie.***).

Hetzelfde kunnen we doen voor de `semijoin` operatie. Als er tussen de intervallen van de heads van beide operanden geen overlap is zal de `semijoin` geen tuples opleveren. Hiervoor hebben we dus van beide operanden de kleinste en de grootste head waarde nodig, ook deze informatie bevindt zich reeds in de fragmentatie database (zie ***Fout! Onbekende schakeloptie-instructie.***).

We kunnen deze operaties helemaal weglaten omdat zij het uiteindelijke resultaat van de originele vraag niet veranderen, dit is het gevolg van de volgende eigenschap van de `union` operatie:

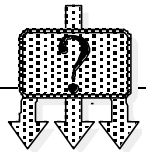
$AB \cup \varnothing = AB$

Ook de Dynamic Loading strategie is gesimuleerd met behulp van de kostenfuncties. Het resultaat is weergegeven in de grafiek in ***Fout! Onbekende schakeloptie-instructie.***. De selectie operatie op processor 3 is nu inderdaad vervallen.



Figuur Fout! Onbekende schakeloptie-instructie. DL uitvoering van de voorbeeldvraag

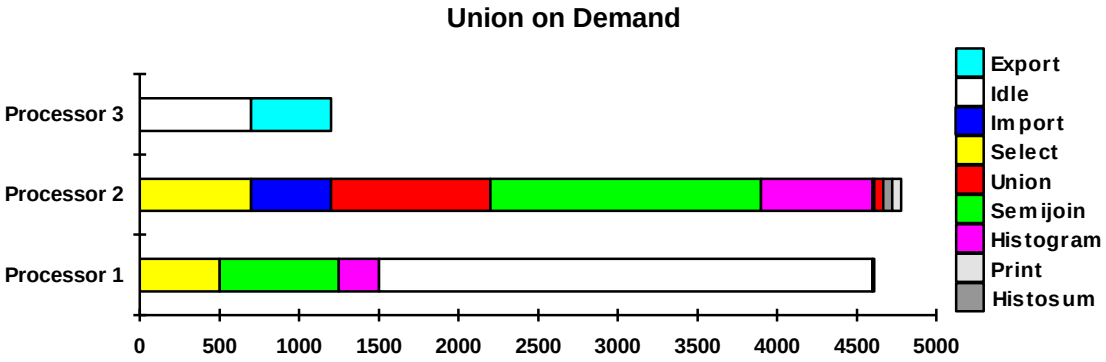
Dit is het resultaat zoals dat door de drie uitvoeringsstrategieën voor gefragmenteerde data kan worden bereikt. We kunnen echter het algoritme nog verder verfijnen. Een eerste stap daartoe wordt in de volgende paragraaf besproken.



9.7 Union on Demand

Tot nu toe hebben we de resultaten van parallelle operaties direct met `union` operaties tot één resultaat samengevoegd. Hierdoor beperken we het parallel uitvoeren van operaties die gebruiken maken van zo'n resultaat. Het resultaat van de selectie uit de voorbeeldvraag komt bijvoorbeeld op de tweede processor terecht. Door zijn geschatte omvang (950 tuples) trekt het vervolgoperaties naar zich toe. Hierdoor komt de `semijoin` operatie ook in zijn geheel op de tweede processor terecht. Het is veel handiger de deelresultaten niet samen te voegen maar het eindresultaat gefragmenteerd in deze deelresultaten over de processors te laten staan. Vervolgoperaties kunnen dan op dezelfde manier geoptimaliseerd worden als met de fragmenten van de basistabellen gebeurt. Het enige moment dat een resultaat samengebracht dient te worden is als er uitvoer naar de gebruiker nodig is, dus in het geval van een `print` operatie. Kortom we voeren alleen `union` operaties uit als daar echt behoefte aan is, vandaar de naam van deze strategie: Union on Demand. Hetzelfde geldt trouwens ook voor operaties die het gevolg zijn van de "verdeel en heers" strategie voor aggregaten, ook deze operaties kunnen worden uitgesteld totdat ze daadwerkelijk nodig zijn.

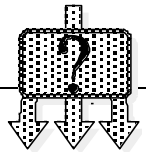
De Union on Demand strategie is ook gesimuleerd en levert de resultaten die staan afgebeeld in de grafiek in ***Fout! Onbekende schakeloptie-instructie.***.



Figuur Fout! Onbekende schakeloptie-instructie. UD uitvoering van de voorbeeldvraag

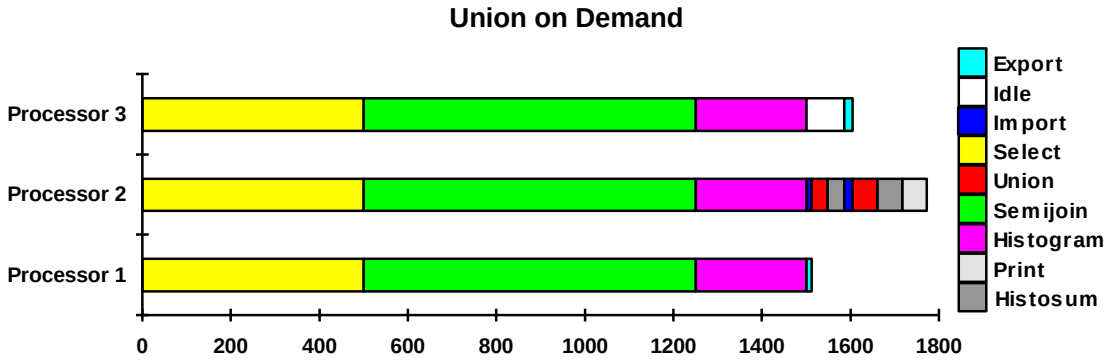
Deze strategie levert een aanzienlijk verbetering ten opzichte van de resultaten van de voorgaande strategieën, de executie tijd is namelijk nagenoeg gehalveerd.

We kunnen het resultaat van deze strategie echter nog verder optimaliseren door de fragmentatie van onze database erop af te stemmen (zie paragraaf 5.3.1). Één van de doelen van het algoritme is zo min mogelijk communicatie tussen de processors te genereren. Als we kijken naar de uitvoering in ***Fout! Onbekende schakeloptie-instructie.***, dan vindt er naast de communicatie benodigd voor het `print` statement nog steeds communicatie voor een `semijoin` operatie plaats. Dit is het gevolg van het feit dat de fragmentatie van `Test_gender` over processor twee en drie is gebaseerd op de waarden in de tail. Het voordeel van deze fragmentatie is dat de `select` operatie bij processor drie is vervallen. Het nadeel is echter dat bij een `semijoin` operatie we communicatie krijgen omdat de fragmenten waarvan er een overlap is tussen de twee head intervallen op één processor bij elkaar moeten worden gebracht.



Fragmentatie van de database op basis van de head zou dus als voordeel hebben dat er minder communicatie wordt gegenereerd. Door al de BATs namelijk te fragmenteren op basis van dezelfde intervallen voor de head waarden worden de fragmenten met dezelfde head waarden ook op één processor geplaatst. Hierdoor zullen de operanden van `semijoin` operaties altijd samenvallen op één processor, waardoor er geen communicatie meer nodig is. Aangezien de `select` operatie relatief goedkoper is dan communicatie tussen processors, zal fragmentatie op basis van de head een betere responstijd opleveren. De fragmentatie van de voorbeelddatabase zal er dan uitzien zoals weergegeven in ***Fout! Onbekende schakeloptie-instructie.***.

Als we de voorbeeld vraag met deze fragmentatie via de Union on Demand strategie uitvoeren krijgen we de resultaten zoals die zijn weergegeven in ***Fout! Onbekende schakeloptie-instructie.***.



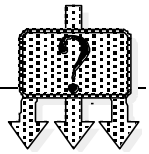
Figuur Fout! Onbekende schakeloptie-instructie. UD uitvoering van de voorbeeldvraag

Halveerde de Union on Demand strategie reeds de responstijd ten opzichte van de Dynamic Loading strategie, deze verbeterde fragmentatie halveert deze de responstijd nogmaals.

Wat kunnen we nu nog verder optimaliseren? Als we kijken naar ***Fout! Onbekende schakeloptie-instructie.*** dan zijn er nog steeds periodes dat een processor niets doet. Processor drie moet wachten voordat hij het resultaat van de `histogram` operatie kan oversturen naar

Processor	BAT	Head		Tail		Cardinaliteit	Ordinaliteit
		minimum	maximum	minimum	maximum		
1	Test_gender	1	500	“f”	“m”	500	2
1	Test_age	1	500	12	63	500	12
2	Test_gender	501	1000	“f”	“m”	500	2
2	Test_age	501	1000	23	56	500	25
3	Test_gender	1001	1500	“f”	“m”	500	2
3	Test_age	1001	1500	33	78	500	19

Figuur Fout! Onbekende schakeloptie-instructie. Verbeterde fragmentatie van de test tabellen

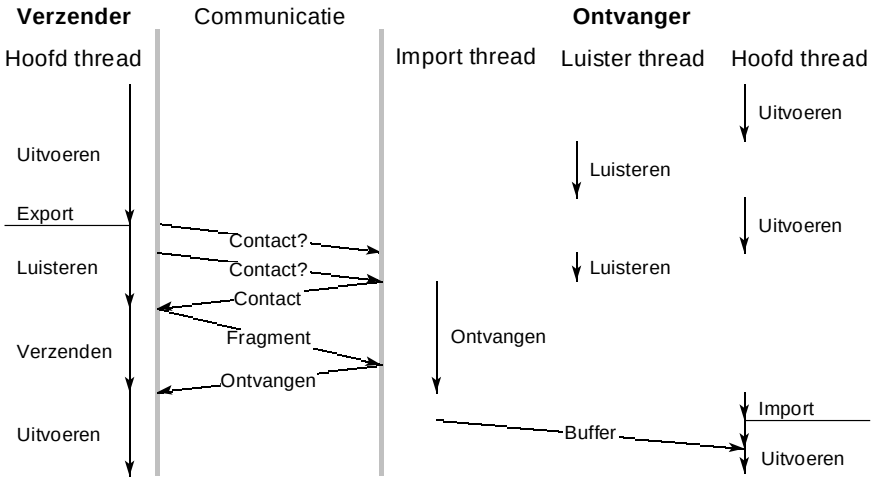


processor twee om uitgeprint te worden. In de volgende paragraaf zullen we deze wachttijd nog verder verminderen.

9.8 Asynchrone communicatie

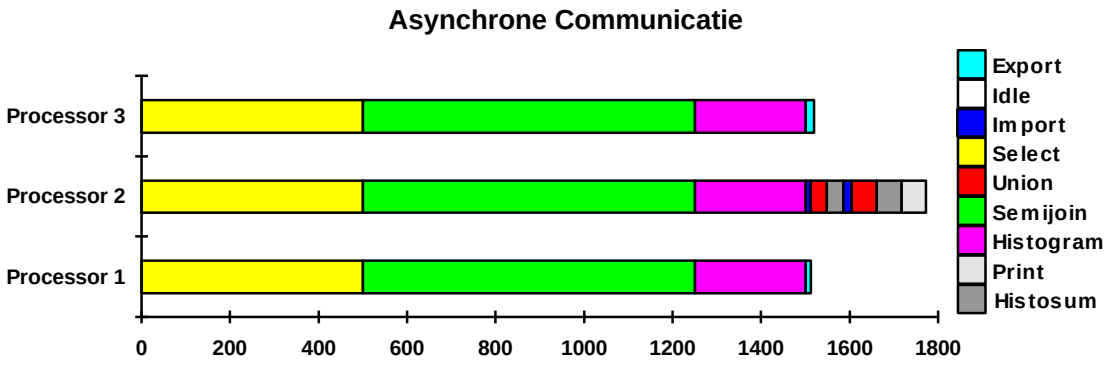
De communicatie vorm waarvan tot nu toe gebruik is gemaakt, is synchrone communicatie. Communicatie tussen de processors vindt pas plaats als de ene processor bij de `export` operatie is aangekomen en de andere bij de bijbehorende `import` operatie. Hierdoor kan het voorkomen dat de ene processor op de ander dient te wachten. Er is echter de mogelijkheid de wachttijd van de processor die de `export` operatie dient uit te voeren tot nagenoeg nul te minimaliseren, namelijk door asynchrone communicatie te gebruiken.

Bij asynchrone communicatie wordt er gebruik gemaakt van threads. Een thread bestaat uit een serie instructies voor een processor. Meerdere threads kunnen het gebruik van één processor delen. Ze krijgen dan om beurten de beschikking over de processor, dit wordt interleaving genoemd. Hierdoor kan de `export` operatie contact krijgen met een “altijd” luisterende thread. Deze luister thread start voor elke processor die contact met hem maakt een aparte `import` thread op. Naar deze `import` thread kan de exporterende processor het fragment oversturen. De `import` thread slaat het ontvangen fragment op in een tijdelijke buffer. Zodra het bijbehorende `import` statement is bereikt wordt het fragment uit de buffer vrijgegeven (zie het diagram in ***Fout! Onbekende schakeloptie-instructie.***).



Figuur Fout! Onbekende schakeloptie-instructie. Asynchrone communicatie

De voorbeeldvraag is nogmaals gesimuleerd en het resultaat is weergegeven in ***Fout! Onbekende schakeloptie-instructie.***. De wachttijd op de derde processor is nu verdwenen.

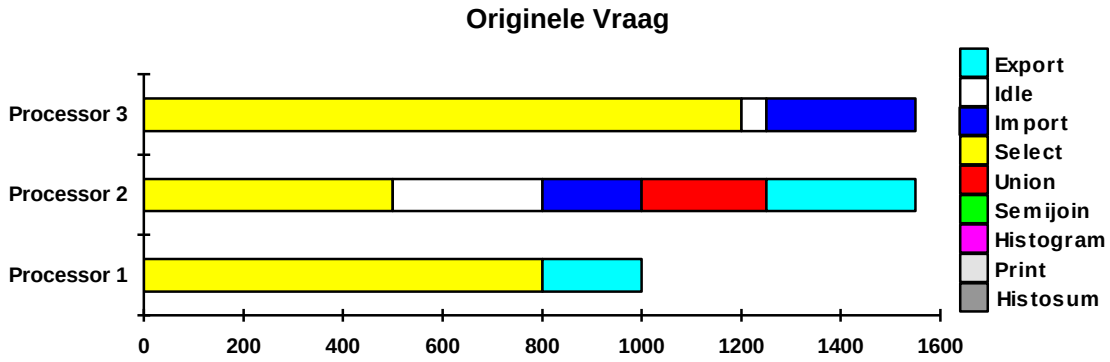


Figuur Fout! Onbekende schakeloptie-instructie. Uitvoering van de voorbeeldvraag met asynchrone communicatie

Via deze vorm van communicatie kan de wachttijd aan de `export` kant vermeden worden. Wachttijd aan de `import` kant is, als de bijbehorende `export` nog niet is bereikt, niet te vermijden. Er is echter wel een methode om deze wachttijd met behulp van de andere communicatie operaties die aan de processor worden toegevoegd te laten verminderen. Deze methode wordt in de volgende paragraaf besproken.

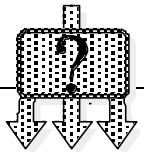
9.9 Subvragen

Tot nu toe hebben we een eenvoudige voorbeeldvraag, met slechts een beperkt aantal operaties, gebruikt. In de praktijk zal één vraag uit veel meer operaties bestaan, zeker als het aantal attributen in de gegevensverzameling groot is. In de grafieken van de voorbeeldvraag was reeds te zien dat de belasting van de verschillende processors uit elkaar gaat lopen, dit effect zal alleen maar groter worden bij een groter aantal operaties. Hierdoor komen ook `import` en `export` operaties verder uit elkaar te liggen, waardoor uiteindelijk de wachttijd ontstaat die we in deze paragraaf willen verminderen. Hoe kunnen we dit probleem nu aanpakken?



Figuur Fout! Onbekende schakeloptie-instructie. Uitvoering van een vraag met wachttijd

In ***Fout! Onbekende schakeloptie-instructie.*** is een situatie weergegeven waarbij de specifieke wachttijd ontstaat. We kunnen de wachttijd opvullen door de `export` operatie op



processor twee naar voren te verplaatsen. Dit kunnen we gerust doen aangezien de `export` operatie door de asynchrone communicatie altijd kan plaatsvinden.

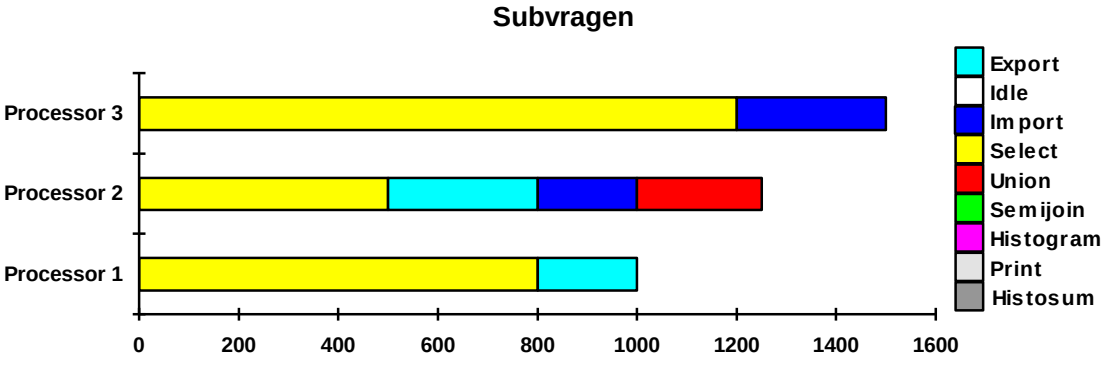
Een probleem ontstaat in de gevallen dat de `export` het resultaat van een voorgaande operatie verzend. Om de `export` operatie zover mogelijk naar voren te kunnen verplaatsen splitsen we de originele vraag daarom op in subvragen waarbinnen de statements onafhankelijk van elkaar zijn, dus de statements gebruiken onderling geen resultaten van statements binnen dezelfde subvraag. In onze voorbeeldvraag zou dit leiden tot vier verschillende subvragen:

```
Subvraag 1: Temp1 := select(Test_gender,"m");
Subvraag 2: t1 := semijoin(Test_age,Temp1);
Subvraag 3: Temp2 := histogram(t1);
Subvraag 4: print(Temp2);
```

In de praktijk zal elke subvraag meer operaties bevatten en kunnen we binnen al deze subvragen de export operaties naar voren schuiven.

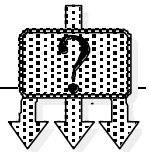
Door de reconstructie regel en de optimalisaties zullen er echter nog statements, zoals `union` en `histosum` operaties, aan een subvraag worden toegevoegd. Het versturen van de resultaten van deze operaties kan niet naar voren worden verplaatst, aangezien deze resultaten dan nog niet bestaan. Wachttijd die ontstaat door deze extra operaties kan, indien ze voor de te verplaatsen `export` operatie plaatsvindt, ook verminderd worden door het verplaatsen. Aangezien elke operatie tot aan de oorspronkelijke plaats van de `export` operatie naar voren schuift, waardoor `import` operaties dichter bij de bijbehorende `export` operatie komen te liggen.

Indien de `export` operatie in ***Fout! Onbekende schakeloptie-instructie.*** geen betrekking heeft op het `union` resultaat, maar op een fragment of een resultaat van een eerdere subvraag gaat de grafiek er als afgebeeld in ***Fout! Onbekende schakeloptie-instructie.*** uitzien.



Figuur Fout! Onbekende schakeloptie-instructie. Uitvoering van een in subvragen verdeelde vraag

Door het verschuiven van de `export` operaties binnen de deelvragen kan veel van de wachttijd worden ingelopen. Het probleem van het verschil in belasting van de processors kan



echter nog steeds blijven bestaan. Hiervoor wordt in de volgende paragraaf een oplossing aangedragen.

9.10 De belasting gelijkmatig verdelen

De belasting van de verschillende processors zal met de aangedragen decompositie van de originele data mining vraag niet gelijk zijn, zeker niet als de fragmentatie van de originele database niet evenwichtig plaats heeft gevonden.

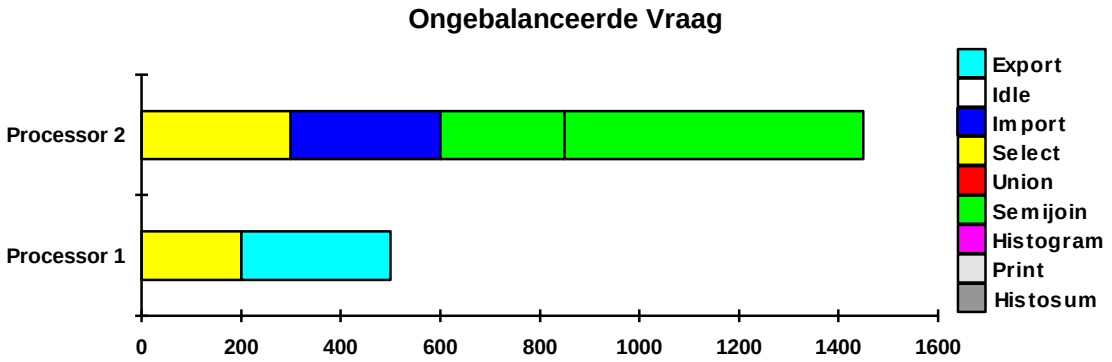
Een voorbeeld van een situatie die op deze manier kan ontstaan is een `semijoin` operatie waarbij de twee deelnemende fragmenten zich op verschillende processors bevinden. Door de huidige allocatie regel wordt het fragment met de kleinste cardinaliteit overgestuurd naar de andere processor, hierdoor kan deze processor veel operaties, met bijbehorende belasting, naar zich toe trekken. In ***Fout! Onbekende schakeloptie-instructie.*** is deze situatie weergegeven.

Afhankelijk van de kosten van communicatie kan het op een gegeven moment interessanter worden om het grotere fragment over te sturen en de `semijoin` operatie op de minder zwaar belaste processor uit te voeren. Hiervoor zullen we dan wel de allocatie regel voor de binaire operaties dienen aan te passen, deze zal rekening dienen te houden met de belasting van de processors.

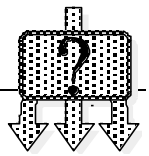
De belasting van een processor kunnen we in de volgende onderdelen uiteenrafelen:

- *gedaan*: belasting van reeds uitgevoerde subvragen;
- *export*: belasting van de naar voren verplaatste `export` operaties;
- *wacht*: wachttijd ontstaan doordat een `export` op een andere processor nog niet bereikt is;
- *import*: belasting van het ontvangen van naar voren verplaatste `export` operaties;
- *union*: belasting afkomstig van operaties toegevoegd door de reconstructie regel en de optimalisaties;
- *werk*: belasting van de overige operaties.

De import en union belasting dienen apart bijgehouden te worden om het inlopen van de wachttijd door het verplaatsen van `export` operaties bij te kunnen houden. Deze volgorde van belasting onderdelen komt overeen met de werkelijke volgorde van de operaties.



Figuur Fout! Onbekende schakeloptie-instructie. Uitvoering van een vraag met een ongebalanceerde belasting



Om deze onderdelen van de totale processor belasting bij te kunnen houden dienen we de kosten van de verschillende operaties te kunnen schatten. Dit kan met behulp van kostenfuncties. Voor de voorbeelden maken we gebruik van de kostenfuncties zoals die in paragraaf 9.3 zijn gespecificeerd.

De binaire allocatieregel dient nu een afweging te maken tussen twee strategieën:

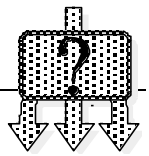
- stuur het fragment van de tweede processor over naar de eerste en voer de operatie op de eerste processor uit;
- stuur het fragment van de eerste processor over naar de tweede en voer daar ook de operatie uit.

Hiervoor heeft de regel informatie nodig over de belasting van de verschillende processors, of een BAT het resultaat is van een operatie in de union belasting van een processor en of een operatie, bijvoorbeeld de `union` of `histosum` operatie, tot de union belasting behoort. Binnen één van de strategieën kunnen we de effecten op de belasting van de processors als volgt bepalen (in pseudo code):

```
Strategy(&BAT1[proc1,load1,union1], &BAT2[proc2,load2,union2], union) {
     als:    niet(union2)
     dan:    als:    (load2.done + load2.export) > load1.comm
             dan:    load1.wait += load2.done + load2.export
                             - load1.comm
             load1.import += costImport(BAT2)
             load2.export += costExport(BAT2)
             load2.wait := maximum(0,load2.wait - costExport(BAT2))
     anders:als:    load2.comm + load2.union > load1.comm + load1.union
             dan:    load1.wait += load2.comm + load2.union
                             - load1.comm - load1.union
             load2.union += costExport(BAT2)
             load1.union += costImport(BAT2)
     als:    union
     dan:    load1.union += costOperation(BAT1,BAT2)
     anders:load1.work += costOperation(BAT1,BAT2)
}
```

De eerste voorwaarde controleert of de over te zenden BAT het resultaat is van een toegevoegde operatie. Zo niet dan kan de `export` operatie verplaatst worden. Als de `export` operatie nu later plaats vindt dan de `import` operatie (`load.comm := load.done + load.export + load.wait + load.import`) wordt er wachttijd aan de `import` kant gegenereerd. Aan de `export` kant geeft het verschuiven van de `export` operatie de mogelijkheid wat van de opgelopen wachttijd in te lopen. Het uitvoeren van een communicatie operatie (`union2` is waar) kan ook wachttijd opleveren. Als laatste worden de kosten van de operatie bij de juiste belasting van de processor toegevoegd. Met deze berekening kunnen we het effect van beide strategieën voorspellen. De eigenlijke allocatie regel maakt nu een afweging tussen deze twee strategieën:

```
BAT[proc] := allocateBinary(&BAT1[proc1,load1,union1],
```



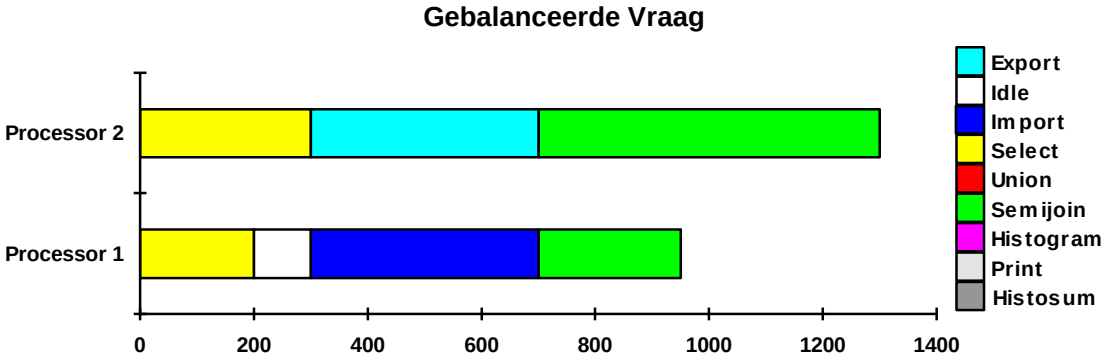
```
                &BAT2[proc2,load2,union2], union) {
      als:   proc1 == proc2
      dan:   proc := proc1
             als:   union
             dan:   load1.union += costOperation(BAT1,BAT2)
             anders:load1.work += costOperation(BAT1,BAT2)
      anders:s1_load1 := load1           //Strategie 1:
             s1_load2 := load2           //BAT2 naar proc1
             Strategy(s1_load1,s1_load2,union)
             s2_load1 := load1           //Strategie 2
             s2_load2 := load2           //BAT1 naar proc2
             Strategy(s2_load2,s2_load1,union)
             als:   maximum(s1_load1.total,s1_load2.total)
                        ✂ maximum(s2_load1.total,s2_load2.total)
             dan:   load1 := s1_load1
                    load2 := s1_load2
                    proc := proc1
             anders:load1 := s2_load1
                    load2 := s2_load2
                    proc := proc2
}
```

Na het berekenen van de effecten van beide strategieën op de belasting van de processors kan de keuze worden gebaseerd op de strategie die de kortste responstijd tot effect heeft.

Om de belasting goed te kunnen bijhouden dienen we ook de unaire allocatie regel aan te passen:

```
BAT[proc,load] := allocateUnary(BAT1[proc1,load1], union) {
     proc := proc1
     als:   union
     dan:   load.union += costOperation(BAT1)
     anders:load.work += costOperation(BAT1)
}
```

Als we deze allocatie regels gebruiken bij het voorbeeld uit ***Fout! Onbekende schakeloptie-instructie.*** ziet de grafiek eruit zoals weergegeven in ***Fout! Onbekende schakeloptie-instructie.***.



Figuur Fout! Onbekende schakeloptie-instructie. Uitvoering van een vraag met een gebalanceerde belasting

De belasting tussen beide processors is nu dichter bij elkaar gekomen. Een nadeel is echter wel dat er nieuwe wachttijd geïntroduceerd wordt, deze kan echter nog door verschuivende `export` statements verminderd worden.

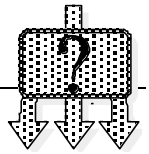
In de volgende paragraaf wordt de laatste wijziging aan het ontwikkelde algoritme besproken.

9.11 Dynamische decompositie

Zoals het algoritme tot nu toe is ontwikkeld wordt een vraag in één keer geoptimaliseerd, gealloceerd en opgesplitst in deelvragen voor de verschillende processors. De grootte van de tussenresultaten worden geschat (zie paragraaf 9.4) en operaties die gebruik maken van deze tussenresultaten worden op basis van deze schattingen geoptimaliseerd en gealloceerd. Deze manier van optimalisatie en allocatie wordt statische query executie genoemd.

Het probleem met deze methode is dat op moment van executie er niet gekeken wordt naar de werkelijke belasting van de processors of de werkelijke samenstelling van de database. Deze beide problemen hebben te maken met de verdeling van de waarden van de verschillende attributen in de database. De executie van de vraag wordt namelijk sterk beïnvloed door de scheefheid van deze verdeling. Wat zijn deze invloeden?

Scheefheid kan binnen een parallel database management systeem op verschillende manieren ontstaan (zie [Walton e.a., 1991]). Allereerst is er de niet uniforme verdeling van de verschillende waarden binnen een attribuut: de ene waarde komt vaker voor dan de andere. Deze scheefheid wordt in het Engels Attribute Value Skew (AVS) genoemd. Deze vorm van scheefheid is niet specifiek voor een parallel database management systeem, maar kan net zo goed voorkomen op een single processor database management systeem. Een selectie van een relatie met AVS kan een veel groter of kleiner resultaat opleveren dan dezelfde selectie op een relatie met een uniforme verdeling. Deze scheefheid is niet te voorkomen door een bepaalde parallelle implementatie, maar is inherent aan de specifieke database.



De tweede vorm van scheefheid is wel specifiek voor parallelle database management systemen en is het gevolg van een ongelijk aantal tuples per processor. Deze scheefheid wordt partition skew genoemd. De verdeling van de tuples over de processors is niet eenvoudig een simpele (uniforme) verdeling van de scheve verdeling van de tuple waarden, maar is afhankelijk van de volgende factoren:

- de AVS in de oorspronkelijke relatie;
- karakteristieken van de gehanteerde fragmentatie methode;
- de uitgevoerde `select` en `semijoin` operaties.

Partition skew kunnen we daarom in vier verschillende soorten verdelen:

- Tuple Placement Skew (TPS): de verdeling van de basis relaties over de processors is ongelijk, bijvoorbeeld doordat de fragmentatie op basis van de tail heeft plaatsgevonden;
- Selectivity Skew (SS): de selectiviteit van een `select` operatie verschilt per processor, zodat de cardinaliteit van het resultaat per processor verschilt;
- Redistribution Skew (RS): voor een `semijoin` operatie worden de tuples opnieuw verdeeld en verschillende processors krijgen bij deze herdistributie verschillende aantallen tuples toegewezen;
- Join Product Skew (JPS): de selectiviteit van een `semijoin` operatie verschilt per processor.

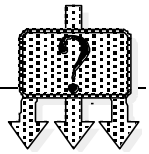
Deze vormen van scheefheid leiden tot verschillende belastingen per processor.

Wat levert dit binnen onze decompositie module voor problemen op? Hiervoor kunnen we de vier vormen van partition skew bekijken, aangezien deze op zijn beurt ondermeer voorkomt uit de AVS.

Allereerst de TPS, deze vorm van scheefheid is al eerder aan de orde gekomen (zie paragraaf 9.7). Fragmentatie op basis van de tail leidt namelijk snel tot deze scheve fragmentatie. Als oplossing voor dit probleem werd het fragmenteren op basis van de head aangedragen. Hiermee kunnen we voor de basis relaties in ieder geval een evenwichtige verdeling van de tuples verkrijgen. Dit wil echter niet zeggen dat we daarmee de AVS uniform over de processors hebben verdeeld. De AVS kan bij het ene fragment groter zijn dan bij de andere. Zo kan een data warehouse bijvoorbeeld elke nacht worden geupdate met de gegevens van de lokale filialen. Hierdoor kunnen er reeksen opeenvolgende tuples ontstaan waarbij bijvoorbeeld de waarde van het lokatie attribuut hetzelfde is. Fragmentatie op basis van de head leidt dan snel tot het toewijzen van deze reeks aan één of enkele processors. Deze processors zullen bij selectie op de specifieke locatie dus een groter resultaat hebben dan andere processors.

De SS is een direct gevolg van de AVS per processor. Dit levert problemen op bij de schattingen van de selectie resultaten. Deze schattingen zijn namelijk gebaseerd op een uniforme verdeling van de attribuut waarden. Laten we in het kort onderzoeken of dit wel een geldige aanname is.

Als we kijken naar het doel van data mining dan zijn we op zoek naar interessante deelverzamelingen in de totale gegevensverzameling. Van deze interessante

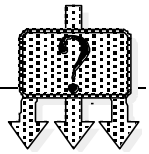


deelverzamelingen willen we een kenmerkende beschrijving hebben. Zo'n kenmerkende beschrijving wordt gevormd door een aaneenschakeling van kenmerkende attribuut waarden. We selecteren dus een interessante deelverzameling en gaan daarbinnen op zoek naar de attribuut waarden die het hoogste onderscheidende vermogen hebben ten opzichte van de rest van de gegevensverzameling. Kortom we zijn op zoek naar de pieken in de frequenties van de waarden. De selecties die we dus uitvoeren om de regels stapsgewijs op te bouwen zullen betrekking hebben op deze pieken. Het baseren van de schattingen op een uniforme verdeling leidt dus tot onjuiste schattingen, bij pieken zijn ze te laag en bij dalen te hoog.

In praktijk blijken databases dan ook geen uniforme waarde verdeling te hebben, maar volgen ze veeleer een Zipf verdeling (zie [Salza e.a., 1989] en [Lynch, 1988]). Via deze Zipf verdeling zouden we de gemiddelde piek frequentie van de waarden kunnen voorspellen. Het probleem van het ene keer een te hoge en een andere keer een te lage schatting blijft echter bestaan, aangezien je niet weet welke waarden een hoge piek frequentie zullen hebben. Daarnaast heeft de Zipf verdeling als parameter de mate van scheefheid nodig, deze is echter ook onbekend. De oplossing voor dit probleem zal verderop in deze paragraaf aan de orde komen.

De volgende twee vormen van scheefheid, RS en JPS, hebben te maken met de `semijoin` operaties. RS heeft te maken met het feit dat bij veel parallelle database management systemen de tuples bij een `semijoin`/`join` operatie opnieuw verdeeld worden. Veel join algoritmes maken namelijk gebruik van hashbuckets. Deze hashbuckets worden aan verschillende processors toegewezen. De tuples worden op basis van een hashfunctie aan een bepaalde hashbucket en daarmee aan een bepaalde processor toegewezen. Door de AVS worden aan sommige hashbuckets veel meer tuples toegewezen dan andere. Op deze manier worden sommige processors veel zwaarder belast dan andere. Oplossingen voor deze vorm van scheefheid proberen deze vorm op tijd te onderkennen en de herverdeling van tuples aan te passen (zie [DeWitt e.a., 1992] en [Dewan e.a., 1994]). Met deze overbelasting van bepaalde hashbuckets hebben we echter niet te maken, omdat de `semijoin` operaties binnen de data mining vraag op basis van de head plaats vinden. Deze head bevat de unieke sleutels van de universele relatie en deze waarden zijn uniform verdeeld, elke waarde komt namelijk maar één keer voor. Dat er toch scheefheid in de belasting van de processors door `semijoin` operaties ontstaat is het gevolg van de andere drie vormen van scheefheid.

JPS heeft, net als SS, te maken met het resultaat van een relationele operatie. Het resultaat van een `semijoin` operatie heeft namelijk ook te maken met AVS. Bij RS hebben we reeds besproken dat de head waarden uniform zijn verdeeld. Betekent dit dat de schattingen ook altijd accuraat zijn? Nee, want de beide relaties kunnen bestaan uit resultaten van eerdere `select` of `semijoin` operaties. Het resultaat van de `semijoin` zal de doorsnede, de head waarden die zowel in de eerste als de tweede relatie voorkomen, van de twee deelverzamelingen van de totale verzameling bevatten. Door de reeds uitgevoerde selecties zullen tuples van beide deelverzamelingen buiten de doorsnede vallen en zal de resulterende cardinaliteit kleiner zijn dan de kleinste cardinaliteit van de twee relaties. De pessimistische schatter geeft dus een maximale cardinaliteit weer.



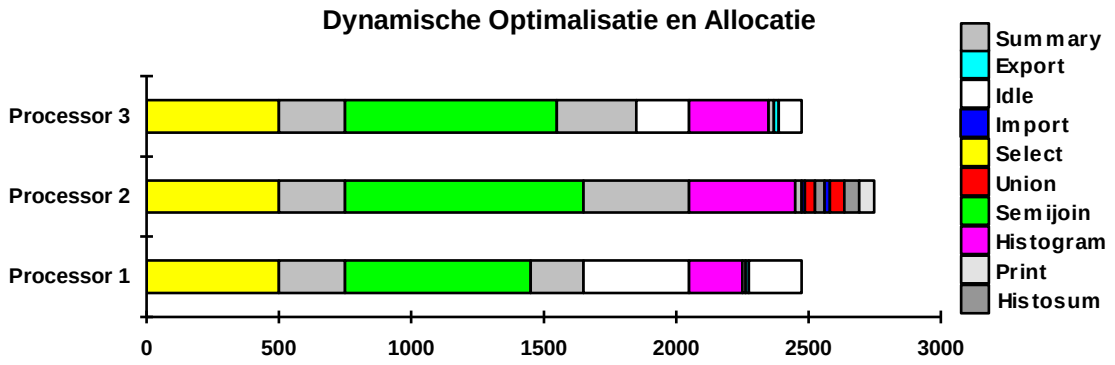
De schatters die we dus hanteren zijn niet bijster betrouwbaar. Het baseren van de schattingen op een andere verdeling levert dezelfde problemen op. Een andere oplossing is de vervolg operaties niet te alloceren op basis van deze schattingen, maar op basis van de werkelijke resultaten. Deze strategie wordt in de database wereld dynamische query executie genoemd. Om de fragmentatie database up-to-date te houden dienen we een extra operatie te introduceren die de gegevens voor de fragmentatie database verzameld: de `summery` operatie. Deze operatie dient de volgende gegevens te verzamelen:

- de minimum head waarde;
- de maximum head waarde;
- de minimum tail waarde;
- de maximum tail waarde;
- de cardinaliteit van het fragment;
- de ordinaliteit (het aantal verschillende waarden) van de tail.

Voor het verzamelen van deze gegevens zal de operatie de hele BAT dienen te doorlopen, de kostenfunctie voor deze operatie luidt dan als volgt:

```
summary(n) := n
```

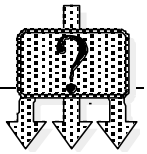
Als we deze operatie toevoegen aan de uitvoering van onze voorbeeldvraag ziet de uitvoering eruit zoals afgebeeld in ***Fout! Onbekende schakeloptie-instructie.***.



Figuur Fout! Onbekende schakeloptie-instructie. Dynamische optimalisatie en allocatie van de voorbeeldvraag

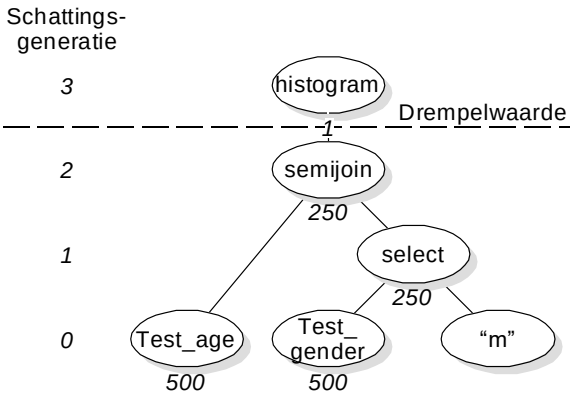
De `summary` operatie wordt nog gealloceerd op basis van de schatting van de cardinaliteit van het resultaat. De volgende operaties kunnen dan echter op basis van de werkelijke gegevens worden gealloceerd. De `select` operaties blijken 200, 400 en 300 tuples op te leveren in plaats van de geschatte 250 tuples per fragment. De kostenberekening van de `semijoin` operatie is daardoor beter, dit kan een doorslaggevend effect hebben op bijvoorbeeld het balanceren van de belasting op de verschillende processors.

We kunnen één subvraag optimaliseren, alloceren en uitvoeren. De volgende subvragen bevat echter statements die afhankelijk zijn van deze voorgaande subvraag. Voordat deze subvraag



dus kan worden geoptimaliseerd en gealloceerd dienen de resultaten van de voorgaande subvraag bekend te zijn. Dit heeft tot gevolg dat de decompositie module pas verder gaat met de volgende subvraag als de processor met de grootste belasting klaar is en de fragmentatie gegevens over zijn resultaten heeft afgeleverd. Dit leidt dus tot synchronisatie van de processors en daarmee tot introductie van wachttijd, wat duidelijk zichtbaar is in ***Fout! Onbekende schakeloptie-instructie.***.

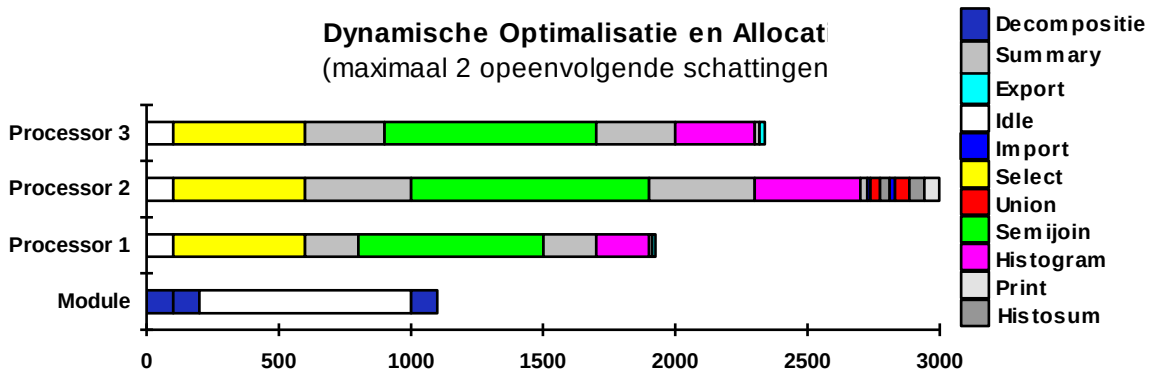
Deze synchronisatie is een ongewenst bijeffect van deze manier van executie. Door toe te staan dat de decompositie module beperkt gebruik maakt van de schattingen kunnen we deze ongewenste synchronisatie beperken. Dit bereiken we doordat de module beperkt wordt in het gebruiken van schattingen die zijn gebaseerd op andere schattingen. We zouden dit een drempelwaarde voor de schattingsgeneratie kunnen noemen. Een maximale schattingsgeneratie van twee betekent dan dat een schatting slechts gebaseerd mag zijn op een schatting die, op zijn beurt, is gebaseerd op werkelijke gegevens (zie ***Fout! Onbekende schakeloptie-instructie.***).



Figuur Fout! Onbekende schakeloptie-instructie. De schattingsgeneratie en de drempelwaarde

Dit heeft tot gevolg dat de module een aantal subvragen vooruit kan werken. Tijdens dit vooruitwerken kunnen de werkelijke fragmentatie gegevens van de resultaten van eerdere subvragen ontvangen worden en kunnen de schattingen op basis van deze werkelijke gegevens bijgewerkt worden, waardoor het aantal schattingsgeneraties verminderd. Als de decompositie module snel genoeg werkt, zodat de processors continue opdrachten krijgen, of de belasting van de processors niet zo zwaar is, zodat werkelijke resultaten op tijd bekend zijn, zal hij op deze manier de processors voorblijven, terwijl de schattingen binnen de opgegeven nauwkeurigheid blijven en zullen de processors alleen aan het eind van de vraag gesynchroniseerd worden. Hiervoor dient de toegestane schattingsgeneratie minimaal twee te zijn. Bij een kleiner aantal wacht de module direct na het verzenden van de deelvragen op de resultaten en krijgen we de situatie zoals weergegeven in ***Fout! Onbekende schakeloptie-instructie.***.

In ***Fout! Onbekende schakeloptie-instructie.*** is de voorbeeldvraag weergegeven bij een dynamische executie waarbij maximaal twee schattingsgeneraties zijn toegestaan. Ook is de verdeling van de belasting van de decompositie module in de grafiek opgenomen.



Figuur Fout! Onbekende schakeloptie-instructie. Beperkte dynamische executie van de voorbeeldvraag

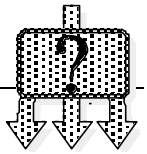
De `select` (eerste schattingsgeneratie) en de `semijoin` (tweede schattingsgeneratie) operaties worden nu op basis van de schattingen gealloceerd, de `histogram` operatie wordt op basis van de werkelijke fragmentatie gegevens bijgewerkt. Om de allocatieregel die de `histogram` operatie een goed beeld van de belasting van de processors te geven dient naast de fragmentatie gegevens ook de werkelijke belasting aan de module gemeld worden, zodat de geschatte belasting kan worden gecorrigeerd. De decompositie module kan het aantal toegestane deelvragen vooruitwerken.

Deze methode werkt alleen als de communicatie tussen de decompositie module en de processors asynchroon plaats vindt. Oftewel de processor kan de fragmentatie gegevens op elk moment kwijt. Als de processor dient te wachten totdat de decompositie module op een vast punt, bijvoorbeeld na het optimaliseren of alloceren van een subvraag, is gearriveerd wordt er alsnog wachttijd gegenereerd en is er kans dat de processors worden gesynchroniseerd (doordat de module de processors één voor één langs gaat om de fragmentatie gegevens op te halen en dan pas de volgende deelvragen kan bepalen). Ook de module dient in staat te zijn asynchroon commando's naar de processors te kunnen sturen, omdat anders de module pas verder kan als de laatste processor zijn volgende commando's heeft ontvangen. En dit zou opnieuw wachttijd kunnen genereren (doordat de module de processors één voor één langs gaat om de commando's af te leveren).

In de voorgaande paragrafen is stapsgewijs het uiteindelijke algoritme ontwikkeld. In de volgende paragraaf zal het totale algoritme op een rijtje worden gezet.

9.12 Het complete algoritme

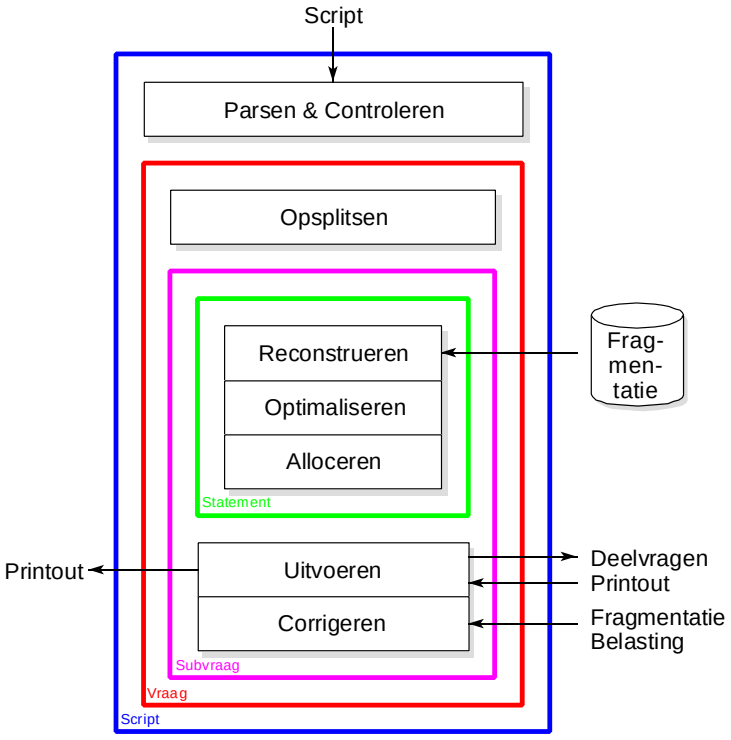
Omdat in de voorgaande paragrafen steeds is ingegaan op één onderdeel van het totale algoritme wordt in deze paragraaf kort het ontwikkelde algoritme beschreven. In ***Fout! Onbekende schakeloptie-instructie.*** is een schematische weergave van het ontwikkelde algoritme afgebeeld. In de afbeelding zijn de verschillende niveaus onderscheiden die in één data mining script kunnen worden onderscheiden. Allereerst is daar het script. Een script bestaat uit een serie MIL statements. Een serie MIL statements kan meerdere vragen omvatten, gescheiden door `commit` operaties. Één vraag wordt opgesplitst in verschillende



deelvragen. Deze deelvragen bestaan op hun beurt uit de individuele statements of operaties. Tussen de verschillende niveaus bevindt dus een één op *n* relatie. In de figuur geven de verschillende kleuren vakken een iteratie over het onderliggende niveau aan. Het opsplitsen van de vraag in subvragen vindt dus per vraag plaats, de optimalisatie en allocatie van de statements vindt weer per subvraag plaats. In de volgende alinea's zullen de verschillende bewerkingsstappen worden toegelicht.

De decompositie methode start met het lezen van een vraag uit het script. De statements in deze vraag worden omgezet in de gepresenteerde boomstructuren (paragraaf 9.3) om zo makkelijker bewerkt te kunnen worden.

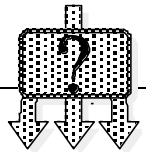
Als een totale vraag is ingelezen wordt deze opgesplitst in subvragen (paragraaf 9.9). Deze subvragen zijn onderling afhankelijk, dat wil zeggen dat een subvraag, op de eerste na, gebruik maakt van het resultaat van een operatie uit een eerdere subvraag.



Figuur Fout! Onbekende schakeloptie-instructie. Schematische weergave decompositie methode

Per subvraag worden de statements nu geoptimaliseerd en gealloceerd. Voordat de optimalisatie kan plaats vinden worden de BATs waarvan het statement gebruik gemaakt via de reconstructie regel hersteld uit de fragmenten (paragraaf 9.4). De operaties voor deze reconstructie regel, bij horizontale fragmentatie `union` operaties, worden in een bushy tree geplaatst, zodat zo optimaal mogelijk gebruik kan worden gemaakt van het parallel uitvoeren van deze operaties.

De optimalisatie bestaat uit het uitvoeren van transformatie regels (paragraaf 9.5). Deze transformatie regels hebben als doel de oorspronkelijke operaties zo laag mogelijk in de boomstructuur te plaatsen. Hierdoor worden de oorspronkelijke operaties opgesplitst in meerdere operaties die parallel kunnen worden uitgevoerd. Voor aggregaten wordt daarbij gebruik gemaakt van een "verdeel en heers" strategie die hetzelfde resultaat heeft. Om vervolg operaties, dus operaties in opvolgende deelvragen die gebruik maken van de resultaten van de operaties in deze deelvraag, ook parallel uit te kunnen voeren, worden de reconstructie operaties die nu bovenin de boom terecht zijn gekomen uitgesteld totdat het totale resultaat

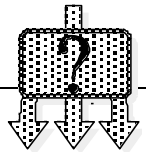


benodigd is, bijvoorbeeld voor een `print` operatie (paragraaf 9.7). Daarnaast worden operaties waarvan op basis van de fragmentatie gegevens bekend is dat zij geen resultaat zullen opleveren, verwijderd (paragraaf 9.6).

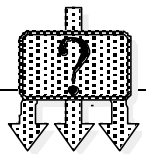
Nu de geoptimaliseerde statements bekend zijn dienen de operaties aan een processor toe te worden gewezen. Hierbij wordt er gebruik gemaakt van allocatie regels (paragraaf 9.10). Deze regels maken, in het geval van een binaire operatie, een keuze tussen twee strategieën: het uitvoeren van de operatie op de ene of op de andere processor. De keuze wordt gemaakt op basis van de gevolgen die beide strategieën hebben op de maximale belasting. Deze gevolgen worden bepaald op basis van de werkelijke cardinaliteiten van fragmenten of schattingen van de resultaten van operaties (paragraaf 9.4) en kostenfuncties die de uiteindelijke belasting berekenen. Degene die de minste maximale belasting, en daarmee de snelste responstijd, oplevert wordt gekozen. Op deze manier wordt er ook naar gestreefd om de belasting tussen de processors te egaliseren. Dit houdt in dat een operatie aan een minder zwaar belaste processor kan worden toegewezen, ook al betekent dit een zwaardere communicatie operatie.

Als al de statements aan een processor zijn toegewezen kunnen ze worden uitgevoerd. Bij het verzenden van de statements naar een processor worden `export` statements zo vroeg mogelijk verzonden (paragraaf 9.8 en 9.9). Hierdoor wordt de wachttijd, die kan worden verkregen doordat `import` operaties op de bijbehorende `export` operatie dienen te wachten, zoveel mogelijk verminderd. Na het verzenden wordt eventuele printout opgevangen en naar de gebruiker doorgesluisd.

In principe kan met behulp van deze beschrijving een data mining vraag reeds worden opgesplitst en parallel uitgevoerd. De schattingen van de cardinaliteiten en de kostenfuncties die aan de basis liggen van de allocatie regel zijn echter niet betrouwbaar (paragraaf 9.11). Deze schatters gaan namelijk uit van een uniforme verdeling van de waarden binnen een BAT. Deze aanname is voor de originele database misschien geldig. Maar deze verdeling wordt door de fragmentatie en de uitgevoerde selecties ongedaan gemaakt. Bij data mining vragen is dit zelfs zeer aannemelijk omdat het doel van de vraag juist is deelverzamelingen in de database bloot te leggen die niet voldoen aan deze uniforme verdeling, men is op zoek naar de pieken en de dalen in de verdeling. De allocatie kan dus beter plaats vinden op basis van de werkelijke resultaten. Om de processors echter niet te veel te synchroniseren kan de gebruiker de maximale schattingsgeneratie opgeven. Deze drempelwaarde geeft aan op hoe veel voorgaande schattingen een schatting mag worden gebaseerd. Naast de werkelijke resultaten dient ook de werkelijke belasting bekend te worden. Zodat volgende operaties beter gealloceerd kunnen worden. Op dit moment van het algoritme kan er gecontroleerd worden of er reeds werkelijke resultaten of processorbelasting bekend zijn. Zo ja, dan kunnen de schattingen worden aangepast en kan het aantal schattingsgeneraties verminderd worden. Indien de drempelwaarde is overschreden dan dient er te worden gewacht totdat de resultaten bekend zijn, op zo'n moment worden de processors dus gesynchroniseerd. Deze synchronisatie kan voorkomen worden als de module snel genoeg werkt, zodat de processors altijd werk hebben, en de belasting van de processors niet zo zwaar is dat de werkelijke resultaten niet beschikbaar zijn voordat de drempelwaarde bereikt is.



Via dit algoritme kunnen we voldoen aan de onderzoeksvraag zoals die is geformuleerd in hoofdstuk 7. De data mining vraag wordt op basis van de fragmentatie gegevens zo optimaal mogelijk verdeeld over de verschillende database servers. In hoofdstuk 10 zal de implementatie van deze decompositie methode besproken worden.



10. Implementatie

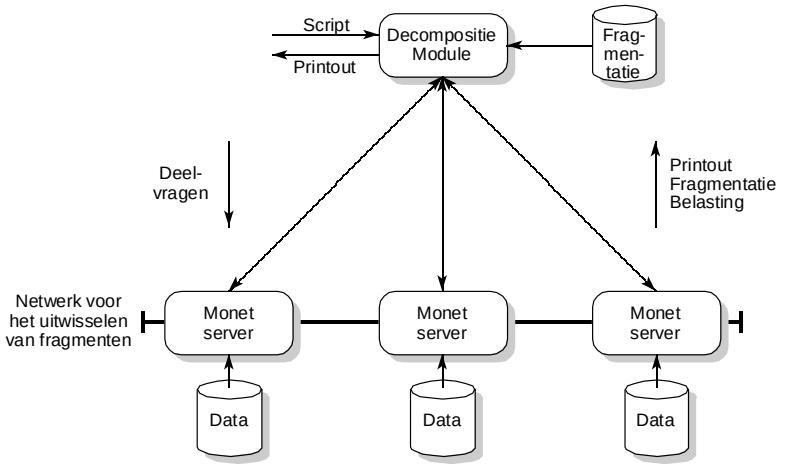
10.1 Inleiding

Om het probleem dat in hoofdstuk 7 is beschreven te kunnen oplossen dient minimaal het in hoofdstuk 9 beschreven algoritme geïmplementeerd te worden. De implementatie dient dan de volgende onderdelen te ondersteunen:

- de Union on Demand uitvoeringsstrategie;
- asynchrone communicatie tussen de servers;
- verschuiven van communicatieoperaties binnen subvragen;
- allocatieregels die zorg dragen voor een gelijkmatige verdeling van de belasting over processors;
- zowel statische als dynamische decompositie.

Via deze implementatie kan dan bepaald worden of het algoritme inderdaad de responstijd van het data mining proces reduceert. De in hoofdstuk 8 beschreven methode om aggregaten parallel te berekenen wordt in deze methode toegepast door de `histogram` operatie te ondersteunen.

In ***Fout! Onbekende schakeloptie-instructie.*** is de uiteindelijk geïmplementeerde positie van de decompositie module afgebeeld. Er zijn twee verschillen ten opzichte van de positie afgebeeld in ***Fout! Onbekende schakeloptie-instructie.***. Allereerst is de koppeling tussen Data Surveyor en de module nog niet uitgewerkt. Daarnaast komt er een extra informatie stroom vanuit de Monet servers, namelijk de fragmentatie en belasting informatie. Deze informatie is benodigd voor de dynamische decompositie methode die in hoofdstuk 9 is ontwikkeld.

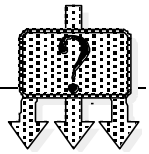


Figuur Fout! Onbekende schakeloptie-instructie. Geïmplementeerde decompositie module

In dit hoofdstuk zullen kort de onderdelen van de implementatie worden geschetst.

10.2 De decompositie module

De decompositie module ontvangt op dit moment het script nog niet van Data Surveyor, maar leest vanuit een bestand het script in. Alle andere communicatie stromen zijn echter volledig geïmplementeerd. De module biedt verschillende instellingen. Zo kan de gebruiker kiezen



tussen de statische of de dynamische decompositie methode. In het eerste geval wordt het hele script op basis van schattingen geoptimaliseerd en gealloceerd. Bij de dynamische methode wordt er gebruik gemaakt van de werkelijke resultaten van operaties om de vervolg operaties te alloceren. Zoals reeds in hoofdstuk 9 is besproken kan het de methode worden toegestaan een beperkt gebruik te maken van de schattingen door een drempelwaarde voor de schattingsgeneratie op te geven. Deze drempelwaarde wordt ook door de implementatie ondersteund. Dit betekent dat de module geregeld controleert of er reeds resultaten bekend zijn, zo ja dan worden de schattingen aangepast. Bij overschrijden van de drempel wordt er gewacht tot alle resultaten bekend zijn. Beide methoden worden ondersteund door implementaties van de reconstructie regel, de transformatie regels, de “verdeel en heers” strategie en de allocatie regels zoals die bij het ontwikkelen van het algoritme zijn besproken.

In de volgende paragraaf zal de uitbreiding aan de kant van de Monet servers worden besproken.

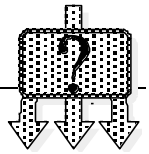
10.3 Uitbreiding functionaliteit van Monet

Eigenlijk zijn er geen wijzigingen nodig aan de kant van de Monet servers. Om de dynamische methode te kunnen ondersteunen is er echter een extra operatie, via een uitbreidingsmodule, toegevoegd. Deze operatie, `summary` genaamd, verzameld in één iteratie over de BAT de benodigde fragmentatie informatie. Deze informatie wordt dan via de standaard connectie tussen de module en de server overgezonden naar de module, zodat daar met behulp van de werkelijke resultaten van een operatie de schattingen kunnen worden aangepast. Naast deze uitbreidingsmodule is er een tweede uitbreiding ontwikkeld om de communicatie tussen de servers te ondersteunen. De communicatie in de totale architectuur van de praktijksituatie zal in de volgende paragraaf behandeld worden.

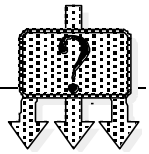
10.4 De communicatie

Alle communicatie stromen in ***Fout! Onbekende schakeloptie-instructie.*** zijn asynchroon geïmplementeerd. In de decompositie module wordt de communicatie met een server bewaakt door een thread. Deze thread buffert zowel de communicatie van als naar de server. De hoofdthread van de module kan operaties voor de server in deze buffer plaatsen, deze operaties worden opgestuurd zodra de server aangeeft operaties te kunnen ontvangen. Aan de andere kant wordt de informatie, printout en fragmentatie gegevens, die door de server wordt opgestuurd ook opgeslagen in een buffer. De hoofdthread van het programma kan deze buffer uitlezen op het moment dat de informatie benodigd is. Hierdoor kunnen zowel de servers als de module in grote mate onafhankelijk van elkaar doorwerken.

Naast communicatie tussen de module en de servers is er ook communicatie benodigd tussen de servers onderling. Deze communicatie is met behulp van een uitbreidingsmodule geïmplementeerd en volgt de werking zoals die is weergegeven in het diagram in ***Fout! Onbekende schakeloptie-instructie.***. De luisterende thread wordt opgestart zodra de server wordt opgestart. De decompositie module verzorgt verder het openen van de connecties en daarmee het opstarten van de `import` threads tussen de servers onderling. Deze implementatie zorgt voor een flinke mate van onafhankelijkheid tussen de servers onderling.



In het volgende hoofdstuk zullen de experimenten die met behulp van deze implementatie zijn uitgevoerd worden beschreven.



11. Experimenten

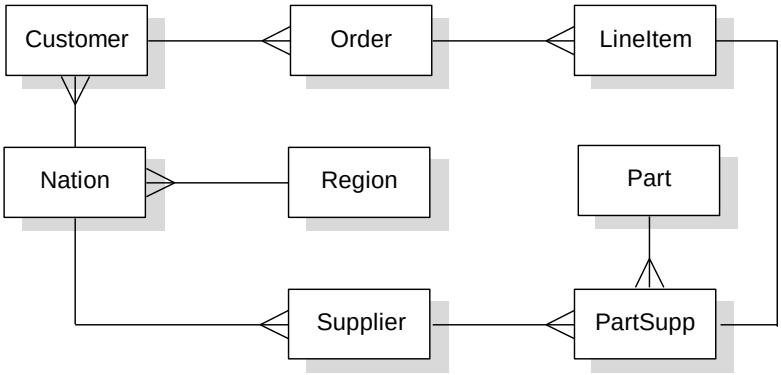
11.1 Inleiding

In dit hoofdstuk zullen de experimenten worden besproken die met behulp van de implementatie van het algoritme zijn uitgevoerd. Het aantal experimenten is beperkt gebleven door instabiliteit van de gehanteerde Monet versie. De uitgevoerde experimenten geven echter een indicatie of het algoritme ook daadwerkelijk een verbetering van de responstijd oplevert. Om te kunnen testen is er echter eerst een testdatabase en een data mining vraag nodig. In de volgende paragraaf zullen deze twee onderdelen worden besproken.

11.2 De testdatabase en de test vraag

Voor het testen is gebruik gemaakt van een gegenereerde database. De generator voor deze database is afkomstig van de Transaction Processing Council (TPC) en genereert een part-supplier database bedoeld voor het testen van beslissing ondersteunende systemen. Het entiteit relatie model van deze database is in ***Fout! Onbekende schakeloptie-instructie.*** weergegeven.

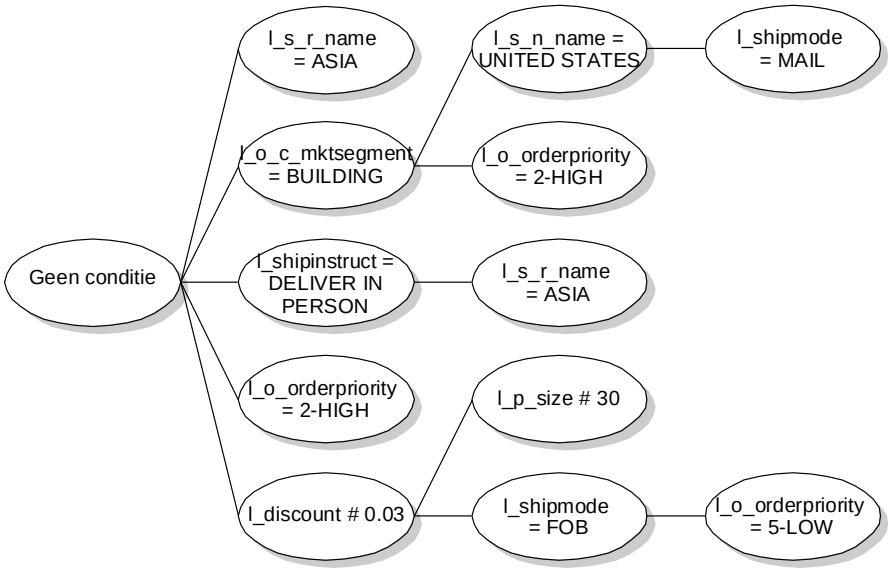
Om voor een data mining sessie gebruik te kunnen maken van deze database is de universele relatie eruit afgeleid. Dit betekent dat er nog maar één tabel over is die alle attributen van de originele database bevat. In dit geval betekent dit dat al de attributen beschikbaar zijn op het niveau van `LineItem`.



Figuur Fout! Onbekende schakeloptie-instructie. ER model TPC-D database

Aangezien de database met behulp van een random generator wordt gegenereerd zijn de waarden van de attributen allemaal uniform verdeeld. Om ook onderzoek te kunnen doen naar de gevolgen van scheefheid wordt de gegenereerde database bewerkt om een schevere verdeling van waarden te verkrijgen. Deze scheefheid is afgestemd op de beslissingsboom die in de volgende alinea wordt besproken.

Om een data mining vraag te kunnen stellen is de database opgesplitst in twee delen: orders die later dan de afgesproken datum zijn geleverd en orders die op tijd binnen kwamen. Het doel van de data mining sessie is nu een karakteristieke beschrijving voor de te late orders te vinden. In ***Fout! Onbekende schakeloptie-instructie.*** is de beslissingsboom die door deze sessie wordt gegenereerd weergegeven.



Figuur Fout! Onbekende schakeloptie-instructie. Beslissingsboom voor de TPC-D database

Om deze beslissingsboom te kunnen genereren zijn er een hele serie database vragen nodig. Het script met deze database vragen wordt gebruikt om de testen uit te voeren.

Aangezien de port van Monet naar de PS/2 nog niet klaar is wordt er als testplatform gebruik gemaakt van een parallelle machine waarin vier UltraSparc processors van 167 Mhz met elk een primair geheugen van 64Mb, ruim gekoppeld zijn. De fragmenten van de originele database bevinden zich op de lokale harde schijf van de verschillende processors.

De (gedeeltelijk) uitgevoerde testen en hun resultaten zullen in de volgende paragrafen besproken worden.

11.3 Speedup

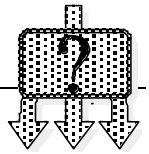
Het allereerste doel van de decompositie module was het verminderen van de responstijd van het data mining proces. Binnen de wereld van parallelle programmatuur en hardware wordt deze versnelling uitgedrukt in de behaalde speedup. Deze speedup wordt als volgt gedefinieerd:

$$speedup = \frac{responstijd\ klein\ systeem}{responstijd\ groot\ systeem}$$

De speedup is lineair als het volgende geldt:

$$speedup = \frac{aantal\ nodes\ groot\ systeem}{aantal\ nodes\ klein\ systeem}$$

Het kleine systeem is de originele situatie van één processor waarop één Monet server draait. Door nu voor een verschillend aantal processors de speedup te bepalen kunnen we kijken of de module in de buurt van de ideale lineaire speedup komt. De originele database van 600.000 tuples wordt hiervoor op basis van de head over een aantal processors verdeeld. De resultaten van de uitgevoerde experimenten zijn in ***Fout! Onbekende schakeloptie-instructie.***



weergegeven. Hierbij is in de parallelle situatie gebruik gemaakt van dynamische decompositie met een toegestane schattingsgeneratie van één.

Er wordt duidelijk speedup behaald. Bij twee processors wordt zelfs meer dan de lineaire speedup behaald. Dit is vooral een gevolg van het feit dat de fragmenten veel kleiner zijn dan de originele database. De originele situatie met één processor krijgt daarom veel eerder te maken met het swappen van geheugen naar harde schijf. In de parallelle situatie kan de hotset in het primaire geheugen blijven wat de responstijd aanmerkelijk verbeterd.

Aantal processors	Responstijd (in ms)	Behaalde speedup
1	328.802	n.v.t.
2	162.201	2,03
3	148.809	2,21

Figuur Fout! Onbekende schakeloptie-instructie. Resultaten bij 600.000 tuples

De stijging van de speedup is echter niet lineair, dit is het gevolg van het feit dat de decompositie module bij een groter aantal processors meer werk te doen krijgt terwijl de processors juist minder belast worden. Dit kan echter nog verbeterd worden door de in de implementatie gehanteerde algoritmes te optimaliseren.

11.4 Scaleup

Naast de speedup is er uit de parallelle informatica wereld nog een andere maat bekend om de prestatie van een algoritme te meten, namelijk de scaleup. Bij bepaling van de scaleup wordt er gekeken of de grootte van het probleem geen invloed heeft op de prestatie van het algoritme. Oftewel als zowel de grootte van het probleem als het aantal beschikbare processors toeneemt, blijft de prestatie van het algoritme dan nog gelijkmatig.
Scaleup wordt als volgt gedefinieerd:

$$scaleup = \frac{responstijd\ kleine\ taak\ op\ een\ klein\ systeem}{responstijd\ \ grote\ taak\ op\ een\ groot\ systeem}$$

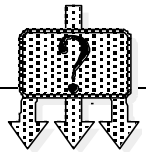
De scaleup is lineair als hij gelijk is aan één.

Om de scaleup van de module te bepalen wordt een steeds grotere database over steeds meer processors verdeeld. Door de problemen met de gehanteerde Monet versie kon dit experiment niet geheel worden voltooid. Enkele eerste resultaten kunnen echter wel gepresenteerd worden.

Naast de experimenten met een database van 600.000 tuples zijn er ook experimenten met een database van 100.000 tuples verdeeld over 2 processors uitgevoerd. In dit laatste geval werd er slechts een marginale speedup bij het gebruik van statische decompositie behaald. In ***Fout! Onbekende schakeloptie-instructie.*** staan de resultaten van de experimenten weergegeven. In dit geval werd er gebruik gemaakt van de originele

Aantal processors	Type decompositie	Responstijd (in ms)	Behaalde speedup
1	n.v.t.	33.223	n.v.t.
2	dynamisch	43.373	0,77
2	statisch	27.663	1,20

Figuur Fout! Onbekende schakeloptie-instructie. Resultaten bij 100.000 tuples

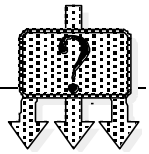


uniforme TPC-D database, dit is de optimale situatie voor de statische decompositie methode die daarom ook goed scoort.

Helaas is de stijging in probleemgrootte (van 100.000 tuples naar 600.000 tuples) niet lineair met de stijging van het aantal processors (van 2 processors naar 3 processors). Daarom kan de scaleup formule niet gehanteerd worden. Als we echter de responstijd van de dynamische decompositie op drie processors op een originele database van 600.000 tuples door vier delen krijgen we een schatting van de responstijd bij een database van 150.000 tuples. Deze schatting levert een responstijd van 37.202 milliseconden op. Hiermee komen we op een scaleup van 1,17. Deze scaleup geeft aan dat een stijging van de probleemgrootte en het aantal processors tot een snellere uitvoering leidt.

Mijn verwachting is dat de scaleup bij grotere databases en grotere aantal processors naar één zal gaan. Deze stijging van 1,17 is een gevolg van het feit dat bij een database grote van 100.000 tuples de belasting van de decompositie module grote invloed heeft op de responstijd. De processors kunnen namelijk niet direct doorwerken maar dienen te wachten tot de volgende subvraag is opgesplitst in deelvragen. Door het kleine aantal tuples slokt deze wachttijd een groot deel van de uiteindelijke responstijd op. De statische decompositie methode heeft hier veel minder last van, de totale rekentijd van de statische decompositie komt echter dichtbij de totale executietijd. Bij een verdere daling van het aantal tuples zal de statische decompositie methode ook meer wachttijd gaan introduceren doordat de module nog bezig is de volgende deelvraag op te splitsen terwijl de servers al klaar zijn met de deelvragen van de voorgaande subvraag.

Deze experimenten omvatten niet alle facetten van het ontwikkelde algoritme. Door problemen met de gehanteerde versie van Monet konden de overige experimenten echter niet worden uitgevoerd. Deze experimenten worden wel beschreven in hoofdstuk 13. In het volgende hoofdstuk zal de conclusie over het resultaat van het totale onderzoek worden getrokken.



12. Conclusies

12.1 Inleiding

In de voorgaande hoofdstukken is een algoritme en een implementatie van dit algoritme ontwikkeld om de responstijd van een data mining sessie te verminderen. In het vorige hoofdstuk zijn de experimenten beschreven die zijn uitgevoerd om de werking van het algoritme en de implementatie te testen. In dit hoofdstuk worden de conclusies, die uit deze resultaten kunnen worden getrokken, besproken.

12.2 Reductie van de responstijd

De centrale vraag van dit onderzoek luidde als volgt:

> *Hoe verdeel ik een data mining vraag zo optimaal mogelijk over verschillende database servers, gegeven een bepaalde fragmentatie van de originele database, en voer ik deze deelvragen uit zodat er ook daadwerkelijk een reductie van de responstijd wordt verkregen?*

Hiervoor is in hoofdstuk 9 een algoritme, dynamische decompositie genaamd, ontwikkeld. Uit de eerste resultaten (hoofdstuk 11) van de implementatie (hoofdstuk 10) van dit algoritme kan de conclusie worden getrokken dat de gewenste reductie van de responstijd daadwerkelijk kan worden verkregen. Uit de resultaten is echter ook gebleken dat er verschillende factoren zijn die invloed hebben op deze reductie of speedup. In de volgende paragraaf zullen deze beperkende factoren kort geresumeerd worden.

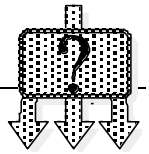
12.3 Grenzen aan de reductie

Deze reductie van de responstijd is afhankelijk van een aantal factoren, namelijk:

- de gehanteerde fragmentatiemethode;
- het aantal tuples in de originele database;
- het aantal in te zetten processors;
- de hoeveelheid primair geheugen per processor.

De eerste factor bepaald in hoeverre er communicatie tussen de processors dient plaats te vinden. Fragmentatie op basis van de head levert de minste hoeveelheid communicatie op. Daarnaast is een evenwichtige verdeling van de originele database in fragmenten ook belangrijk. Onevenwichtige fragmentatie leidt tot scheve belasting van de processors en daarmee tot een langere responstijd.

De tweede factor bepaald of de responstijd van uitvoering in de oorspronkelijke situatie nog wel verbeterd kan worden. Onder de 100.000 tuples gaat de tijd die de decompositie module nodig heeft om de originele vraag te verdelen over de servers de responstijd overheersen en heeft de optimalisatie uiteindelijk een negatieve invloed op de responstijd.



Door meer processors in te zetten wordt de responstijd ook beter. Deze stijging is speedup zal echter stagneren op het moment dat de belasting van de processors zo klein wordt dat de rekentijd van de decompositie module de responstijd weer gaat overheersen. Hoe meer processors er worden ingezet, hoeveel meer situaties de module dient te analyseren waardoor de rekentijd van de module ook toeneemt. Deze wisselwerking zal uiteindelijk leiden tot een plafond aan het aantal doeltreffend inzetbare processors.

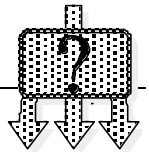
Naast deze factoren heeft de hoeveelheid primair geheugen van de processors ook invloed. Kan dit geheugen de fragmenten in zijn geheel bevatten dan zal een goede responstijd behaald worden. Als het geheugen echter te klein is dan wordt er geswapt en dit heeft een grote invloed op de responstijd. Het inzetten van meer processors en daarmee tevens een kleinere fragment grote kan dit probleem oplossen, zolang de hiervoor besproken grenzen niet worden bereikt.

De conclusie uit de vorige paragraaf wordt door deze factoren beperkt. Een duidelijke vaststelling van de grenzen is nog afhankelijk van verder onderzoek.

12.4 Statisch of dynamisch?

Bij het ontwikkelen van het algoritme in hoofdstuk 9 is een onderscheid gemaakt tussen statische en dynamische decompositie. Beide methoden blijken in staat te zijn reductie van de responstijd op te leveren. Een duidelijke keuze tussen beide methoden kan helaas nog niet worden gemaakt als gevolg van enkele technische problemen.

De dynamische decompositie zal echter pas zin hebben als de allocatieregels in staat zijn de scheve belasting gelijk te trekken. Bij fragmentatie op basis van de head is deze mogelijkheid slechts beperkt, namelijk alleen bij de `union` operaties benodigd voor `print` operaties. Hoe de fragmentatie op basis van de head kan worden aangepast om meer allocatie toe te staan is één van de onderwerpen die in het volgende hoofdstuk aan de orde komen. In het volgende hoofdstuk zullen namelijk mogelijke verbeteringen en experimenten geïntroduceerd worden.



13. Verder Onderzoek

13.1 Inleiding

De dynamische decompositie methode die in dit onderzoek is ontwikkeld en geïmplementeerd is veeleer een raamwerk dan een afgeronde methode. Veel van de onderdelen kunnen namelijk nog verbeterd worden. In dit hoofdstuk zullen enkele van deze verbeteringen aangestipt worden.

Daarnaast konden niet alle geplande experimenten binnen het onderzoekstermijn worden afgerond. Deze geplande experimenten zullen in dit hoofdstuk ook kort beschreven worden.

13.2 Verbeteringen

13.2.1 Verbeterde schattingen

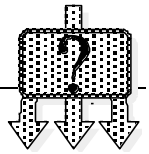
Allereerst is daar het model van de werkelijkheid dat door het algoritme gehanteerd wordt. Dit algoritme is afhankelijk van twee soorten schattingen, namelijk schattingen van de cardinaliteiten van de resultaten van operaties en de kosten van deze operaties. Zoals reeds in hoofdstuk 9 aan de orde is geweest zijn deze schattingen niet accuraat, daarom is er ook gekozen voor een dynamische in plaats van een statische aanpak. Deze beide schattingen kunnen echter worden geoptimaliseerd. Bij de cardinaliteitsschattingen kan dit bijvoorbeeld door specifieke op een bepaalde praktijksituatie afgestemde schattingsfuncties door de gebruiker te laten ontwikkelen en deze als modules te laten laden. De kostenfuncties kunnen ook uitgebreider. Zo speelt de aanwezigheid van zoekversnellers, zoals een index, een grote rol bij duur van een operatie. Een probleem is echter dat de module niet weet op welk fragment een index bestaat. Deze informatie kan echter bij de fragmentatie gegevens worden opgenomen. Door een kosten algebra te ontwikkelen kan de module dan bepalen of een index door propageerd of niet.

13.2.2 Dynamische schattingen

Naast het verbeteren van de functies zelf is er ook de mogelijkheid de functies runtime aan te passen. Dit zou betekenen dat de terug gemelde cardinaliteiten en responstijden door de module worden gebruikt om de schattingsfuncties voor zowel de cardinaliteiten als de kosten te corrigeren. Naar enkele stappen in het algoritme zouden deze schattingen dan beter kunnen zijn. Een andere mogelijkheid zou zijn een speciale vraag uit te voeren die als het ware een leerset van selectiviteitsfactoren oplevert, aan de hand waarvan de initiële schattingsfuncties worden aangepast. Deze aangepaste functies worden dan gebruikt tijdens de uiteindelijke data mining sessie.

13.2.3 Globale allocatieregels

Een ander probleem dat in het huidige algoritme bestaat is dat een processor helemaal van deelname aan een vraag kan worden uitgesloten. Dit vindt plaats als een fragment geen enkele tuple bevat die van belang is voor een classificatieregel. Een uitbreiding van de allocatieregels



zou kunnen zijn dat niet alleen naar de processors die een fragment bevatten dat in de behandelde statement een rol speelt wordt gekeken maar naar alle processors.

13.2.4 Ruime en gesloten koppeling

Voor een snellere responstijd bij grote fragmenten kan nog onderzocht worden hoe Monet servers die op hun beurt beschikken over meerdere processors via een gesloten koppeling kunnen worden ingezet. De deelvragen bestaan namelijk voor een deel uit onafhankelijke statements. Deze statements kunnen op hun beurt parallel worden uitgevoerd (zoals beschreven in hoofdstuk 6).

13.2.5 Parallel sorteren

In hoofdstuk 8 is reeds aangegeven dat er ook behoefte bestaat uit een sorteer methode waarbij de gesorteerde resultaat uiteindelijk ook gefragmenteerd over de verschillende processors achterblijft. Deze sorteer operatie zal uit lokale sorteer operaties en globale uitwisselingsoperaties dienen te bestaan.

13.2.6 Beperken van de summary operatie

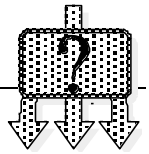
De `summary` operatie voor dynamische decompositie voegt een extra belasting aan de uitvoering van de vraag toe. Deze belasting moet tot het minimum beperkt worden. Dit kan bijvoorbeeld door een extra drempelwaarde in te stellen. Deze drempelwaarde geeft een maximale afwijking tussen de geschatte en werkelijke cardinaliteit aan. Als de drempelwaarde overschreden wordt gaat de `summary` operatie door met het verzamelen van de overige fragmentatiegegevens, in het andere geval worden de geschatte gegevens gehandhaafd.

13.2.7 Beperken van de optimalisatie

Verder kan de cardinaliteit van de resultaten ook invloed uitoefenen op het wel of niet uitvoeren van de optimalisaties. Bij kleine cardinaliteiten heeft deze optimalisatie nog maar weinig zin. Ook voor de optimalisatie kan dus een drempelwaarde worden ingesteld.

13.2.8 Optimalisatie van de fragmentatie voor allocatie

In het voorgaande hoofdstuk werd ook reeds aangestipt dat de allocatieregels bij fragmentatie op de head niet veel mogelijkheden hebben om de allocatie van de operaties te veranderen. Dit is een gevolg van het feit dat de operanden van binaire operaties zich in dat geval altijd op dezelfde processor bevinden. Er kan worden besloten om meerdere processors één interval te laten beheren. De belasting op dit head interval kan dan door de allocatieregel worden verdeeld. Door de binaire operaties in de data mining vragen te analyseren kan er bepaald worden welke interval fragmenten bij elkaar geplaatst dienen te worden en welke juist op verschillende processors moeten worden geplaatst om correctie door de allocatieregels mogelijk te maken.



13.3 Experimenten

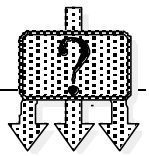
13.3.1 Statisch of dynamisch?

Bij het ontwikkelen van het algoritme zijn twee vormen van executie aan de orde gekomen, dynamische en statische decompositie. Beide methoden zijn met behulp van de implementatie mogelijk. Het doel van dit experiment is te controleren of de scheve verdeling van waarden inderdaad grote invloed heeft op de optimalisatie en de allocatie. Hiervoor wordt de testvraag uitgevoerd met zowel de statische als de dynamische, met een afnemend aantal toegestane opeenvolgende schattingen, decompositie methode uitgevoerd op databases met verschillende maten van scheefheid. De verwachting is dat hoe dichter de waarde verdeling van de attributen in de buurt komt van de uniforme verdeling de statische decompositie methode beter zal gaan presteren.

13.3.2 Fragmentatie en gelijkmatige belasting

In hoofdstuk 9 is reeds aan de orde gekomen dat fragmentatie op basis van de tail extra communicatie zal introduceren. Het effect van de keuze voor een bepaalde fragmentatie, head of tail, wordt via deze experimenten getest. Hierbij is het interessant om het effect van de allocatieregels te controleren. De fragmentatie op basis van de tail geeft deze regels namelijk de kans om de belasting op meerdere posities glad te trekken. Fragmentatie op basis van de tail zal daarom in principe tot een gelijkmatigere verdeling van de belasting dienen te leiden.

In het volgende en laatste hoofdstuk wordt de gehanteerde literatuur opgesomd.



14. Literatuur

De volgende literatuur is gebruikt bij het maken van deze scriptie:

[Adriaans e.a., 1996] Pieter Adriaans & Dolf Zantige, 1996, *Data Mining*, Addison Wesley Longman Limited

[Agrawal e.a., 1993] R. Agrawal, T. Imielinski & A. Swami, 1993, 'Mining association rules between sets of items in large databases', *Proceedings of the 1993 International Conference on Management of Data*

[Berg, van den, 1994] Carel Arie van den Berg, 1994, *Dynamic Query Processing in a Parallel Object-Oriented Database System*, Proefschrift, Universiteit van Twente

[Boncz e.a., 1994] Peter A. Boncz & Martin L. Kersten, 1994, *Monet: An Impressionist Sketch of an Advanced Database System*

[Date, 1995] C.J. Date, 1995, *An Introduction to Database Systems*, Addison Wesley

[de Waard, 1992] Huub de Waard, 1992, *Parallelle verwerking*, Kluwer Technische Boeken b.v.

[Dewan e.a., 1994] Hasanat M. Dewan & Mauricio Hernández, 1994, 'Predictive Dynamic Load Balancing of Parallel Hash-Joins over Heterogenous Processors in the Presence of Data Skew', *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems*

[DeWitt e.a., 1992] David J. DeWitt, Jeffrey F. Naughtong, Donovan A. Schneider & S. Seshadri, 1992, 'Practical Skew Handling in Parallel Joins', *Proceedings of the 18th VLDB Conference*, Vancouver, Britisch Columbia, Canada

[Dicker e.a., 1990] Mat Dicker, Karel Lemmer & Henk Plessius, 1990, *Gegevensmodellering met het Entiteit-Relatie model en de taal Gordas*, Kluwer Bedrijfswetenschappen

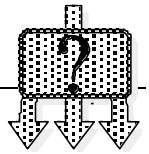
[Duncan, 1990] Ralph Duncan, 1990, 'Een overzicht van de architecturen voor parallelle computers', *Informatie*, juni 1991 (vertaling van artikel uit Computer februari 1990)

[Flynn, 1966] M.J. Flynn, 1966, 'Very high speed computing systems', *Proceedings IEEE*, jaargang 54, blz. 1901-1909

[Gray e.a., 1995] Jim Gray, Adam Bosworth, Andrew Layman & Hamid Pirahesh, 1995, *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals*, Microsoft Research Technical Report MSR-TR-95-22

[Hamers, 1996] Rien Hamers, 1996, 'Bestuurlijke informatievoorziening met EIS', *Informatie*, december 1996, blz. 44-50

[Hattem e.a., 1996] Paul van Hattem & Menzo Windhouwer, 1996, *Database Systemen - Parallelle Database Management Systemen*, APR scriptie, Universiteit van Amsterdam



[Hertz, 1991] J. Hertz, 1991, *Introduction to the Theory of Neural Computing*, Addison Wesley

[Hill, 1994] Jonathan M.D. Hill, 1994, *An introduction to the data-parallel paradigm*, Technical Report QMW-688, Queen Mary and Westfield College

[Holsheimer e.a., 1994] Marcel Holsheimer & Arno Siebes, 1994, *Data Mining The Search for Knowledge in Databases*, CWI Rapport CS-R9406

[Holsheimer e.a., 1995] Marcel Holsheimer & Martin L. Kersten, 1995, 'Architectural Support for Data Mining', *Int. Workshop on Knowledge Discovery in Databases*, Seattle

[Hwang, 1993] Kai Hwang, 1993, *Advanced computer architecture*, McGraw-Hill

[Inmon, 1996] W.H. Inmon, 1996, *Building the Data Warehouse - Second Edition*, New York, Wiley

[Kersten e.a., 1995] Martin L. Kersten & Marcel Holsheimer, 1995, *On the Symbiosis of a Data Mining Environment and a DBMS*, CWI

[Kersten, 1994] Martin L. Kersten, 1994, *Monet Reference Manual*

[Kersten, 1995] Martin Kersten, 1995, *Sheets Intelligente Databases*, Universiteit van Amsterdam

[Khoshafian e.a., 1988] Setrag Khoshafian & Patrick Valduriez, 1988, 'Parallel Execution Strategies for Declustered Databases', *Database and Knowledge Base machines*, blz. 458-471, Kluwer Academic

[Kröse e.a., 1995] Ben J.A. Kröse & Patrick van der Smagt, 1995, *An introduction to Neural Networks*, Reader, Universiteit van Amsterdam

[Lynch, 1988] Clifford A. Lynch, 1988, 'Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distributions of Column Values', *Proceedings of the 14th VLDB Conference*, Los Angeles, California

[McKenzie e.a., 1992] L. Edwin McKenzie & Richard T. Snodgrass, 1992, 'Evaluation of Relational Algebras Incorporating the Time Dimesion in Databases', *ACM Computing Survey*

[Michalewicz, 1989] Z. Michalewicz, 1989, *Genetic Algorithms + Data Structures = Evolution Programs*, New York, Springer-Verlag

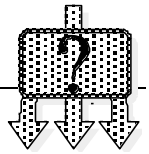
[Özsu e.a., 1991] M.Tamer Özsu & Patrick Valduriez, 1991, *Principles of Distributed Database Systems*, Prentice Hall

[Quinlan, 1986] J. Ross Quinlan, 1986, 'Induction of decision trees', *Machine Learning 1*

[Salza e.a., 1989] Silvio Salza & Mario Terranova, 1989, 'Evaluating the Size of Queries on Relational Databases with non Uniform Distribution and Stochastic Dependence', *Proceedings of the 1989 ACM SIGMOD conference*, Portland, Oregon

[Siebes, 1995] Arno Siebes, 1995, *Data Mining and KDD: an introduction*, Sheets Intelligente Databases, Universiteit van Amsterdam

[Sloot e.a., 1995] P.M.A. Sloot & A.G. Hoekstra, 1995, *Parallel Scientific Computing*, Reader, Universiteit van Amsterdam



[Stubbs e.a., 1989] Daniel F. Stubbs & Neil W. Webre, 1989, *Data structures with abstract data types and Pascal - Second Edition*, Brooks/Cole

[Tannenbaum, 1990] A.S. Tanenbaum, 1990, *Structured computer organization*, Prentice-Hall International

[Walton e.a., 1991] Christopher B. Walton, Alfred G. Dale & Roy M. Jenevein, 1991, 'A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins', *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain

[Wilschut, 1993] Anita Wilschut, 1993, *Parallel Query Execution in a Main-Memory Database System*, Proefschrift, Universiteit van Twente