

Een analyse van de
boomwijzigingsafstand voor
sjablooninductie van
HTML-bomen

Erik Borra

scriptie ingeleverd voor het behalen van de graad

Master of Science

in de Kunstmatige Intelligentie

bij de Universiteit van Amsterdam,

Nederland

31 augustus 2006

Inhoudsopgave

Ten geleide	i
Samenvatting	i
1 Inleiding	3
1.1 Aanleiding tot deze scriptie	3
1.2 Automatische sjablooninductie	5
2 Herhaling in de structuur van HTML-bomen	8
2.1 Herhaling	8
2.2 HTML-bomen	9
2.3 De deelboomstructuur	10
3 Wijzigingsafstanden	12
3.1 De wijzigingsafstand voor strings	12
3.2 De boomwijzigingsafstand	15
3.2.1 Definities	15
3.2.2 Wijzigingen en afbeeldingen	17
3.2.3 Complexiteit	19
3.3 Beperkte top-down boomwijzigingsafstand	19
3.4 Samenvatting	22
4 De veralgemening van HTML-pagina's	23
4.1 Sjablonen	23
4.1.1 Jokers	24
4.1.2 Het bouwen van sjablonen	26
4.2 Prijsfuncties	27
4.2.1 Inertie	27
4.2.2 Vervangingen mogen niet altijd	28
4.3 Nabewerkingen	29
4.4 Extractie	30
4.5 Samenvatting	32
5 Een benadering van de beperkte top-down boomwijzigingsafstand	33
5.1 De hoge prijs voor de berekening van de boomwijzigingsafstand	33
5.2 Een algoritme dat de beperkte top-down afbeelding benadert	34

5.3	De prijsfunctie voor de benadering	37
5.4	Complexiteit	41
5.5	Samenvatting	42
6	Experimentele resultaten	43
6.1	Trainings- en testverzamelingen	44
6.2	De vergelijking van de snelheid tussen de beperkte top-down boom- wijzigingsafstand en mijn algoritme	46
6.3	Aantal leervoorbeelden	46
6.4	Precisie	47
6.5	Nabewerkingen	51
7	Gerelateerd werk en toepassingen	53
7.1	Gerelateerd werk	53
7.2	Toepassingen	55
8	Conclusie en discussie	58
A	Woordenlijst	61
B	Uitleg voor digibeten	63
C	English abstract	65
	Bronvermelding	66

Lijst van figuren

3.1	De pseudocode voor de berekening van de wijzigingsafstand door middel van dynamisch programmeren.	14
3.2	Voorbeeld van een matrix gecreëerd voor de berekening van de wijzigingsafstand tussen van $T=bcdeffghixkl$ en $P=abcdefghijkl$. De prijs van elke wijziging is hier 1. De optimale afbeelding is onderstreept. . . .	14
3.3	Voorbeeld van een afbeelding waarbij de wijzigingen voor de omzetting van B_1 in B_2 de invoeging van B, de verwijdering van D en de vervanging van C door G zijn.	18
3.4	Voorbeeld van een beperkte top-down afbeelding.	21
4.1	Een voorbeeld van het sjabloon S ; de veralgemening van de twee bomen B_1 en B_2	25
4.2	Sjabloon S en Boom B.	31
5.1	De pseudocode voor de wijzigingsafstand in bomen.	35
5.2	Gewenst sjabloon S_g en sjabloon met de nieuwe prijsfunctie S_1	39
5.3	Sjablonen gecreëerd met prijsfunctie 1 & 2.	40
6.1	De veralgemening van twee bomen B_1 en B_2 en de toepassing van het resulterende sjabloon S op een derde boom B_3	45
6.2	Het effect van eerst snoeien en dan Reis' nabewerkingsstap.	52

Lijst van tabellen

6.1	Vergelijking van de gemiddelde leersnelheid benodigd voor de veralgemening van tien keer twee HTML-bomen, in seconden.	47
6.2	Procentuele vergelijking van het aantal leervoorbeelden dat uitlijnt met een sjabloon gecreëerd na het leren op x voorbeelden.	48
6.3	Het gemiddelde aantal geëxtraheerde deelbomen voor sjablonen gecreëerd met de beperkte top-down boomwijzigingsafstand.	49
6.4	Het gemiddelde aantal geëxtraheerde deelbomen voor sjablonen gecreëerd met mijn benadering van de beperkte top-down boomwijzigingsafstand.	50

Ten geleide

Het voor U liggende werkstuk is in het Nederlands geschreven. Vandaag de dag kan dat verwondering wekken, maar ik heb er heel bewust voor gekozen. Ik ben er niet enkel van overtuigd dat ik mij in mijn moedertaal, het Nederlands, het duidelijkst kan uitdrukken, maar ook ben ik van mening dat in mijn vakgebied, de Kunstmatige Intelligentie, veel te weinig in het Nederlands wordt gepubliceerd. Ik hoop met deze scriptie hieraan enigszins te verhelpen. Omdat er heel weinig Nederlandse teksten bestaan over kunstmatige intelligentie en informatica liggen niet alle Nederlandse begrippen in deze tekst voor de hand. Daarom vindt U een vertalende woordenlijst op pagina 61.

Tegelijkertijd wil ik van de gelegenheid gebruik maken een aantal mensen te bedanken. In de eerste plaats mijn begeleider, Dr. Maarten van Someren, die mijn eindwerk heeft begeleid. Verder speciale dank aan Samson de Jager, wiens waardevolle opmerkingen mij hebben geholpen bij de laatste afwerking van mijn scriptie, aan mijn moeder Marguerite Lely, die gezorgd heeft voor de taalkundige laatste hand, en aan Dr. Richard Rogers voor de leuke en interessante werkomgeving waar het idee voor deze scriptie ontstaan is.

Samenvatting

In deze scriptie wordt onderzocht hoe de boomwijzigingsafstand (tree-edit-distance) gebruikt kan worden voor het probleem van wrapper-inductie. De boomwijzigingsafstand wordt gebruikt om een zo goedkoop mogelijke afbeelding te vinden tussen twee boomrepresentaties van HTML-pagina's. Met behulp van deze afbeelding kan een sjabloon geleerd worden waarin alleen nog die delen staan die de bomen gemeenschappelijk hebben. De delen die instantiespecifiek zijn worden gerepresenteerd door jokerknopen. Wat overblijft is de meest specifieke veralgemening van de pagina's die in staat is andere instanties van het zelfde semantische type te herkennen. Het sjabloon kan dan gebruikt worden om instantiespecifieke informatie uit gelijkaardige pagina's te extraheren.

In deze scriptie wordt aangetoond dat het domein van automatisch gegenereerde HTML-bomen een aantal eigenschappen heeft waardoor de boomwijzigingsafstand benaderd en sneller berekend kan worden. Ook worden meerdere nabewerkingsstappen onderzocht die de sjablonen compacter maken en behoeden voor overfit. Het blijkt dat het gebruik van snoeimethodes altijd goede resultaten oplevert.

Hoofdstuk 1

Inleiding

1.1 Aanleiding tot deze scriptie

Inherent aan het web zijn de hyperlinks die een verwijzing naar een andere pagina of een deel daarvan bevatten; het zijn verbindingen tussen brokjes informatie. Tim Berners-Lee, een van de grondleggers van het huidige Wereldwijde Web, beschouwde de hyperlink als de mogelijkheid om elk stukje informatie met elk ander stukje informatie te verbinden [1].

Volgens Marres & Rogers [2] kan op twee interessante manieren naar hyperlinks gekeken worden: je kunt kijken welke hyperlinks een bepaalde gebruiker of een groep van gebruikers volgt (bijvoorbeeld via logmining); je kunt ook bekijken wat webmasters de surfers willen laten weten — door de ‘selectie’ van relevante websites door middel van een hyperlink (hyperlinkanalyse). Beide zienswijzen gaan ervan uit dat een goede zoektocht op het web gestuurd wordt door sociale verbanden. De eerste manier zoekt deze verbanden in de gedeelde interesses van gebruikers; de tweede lokaliseert ze in het al dan niet gemeenschappelijke linkgedrag van organisaties.

Deze laatste interpretatie wordt gebruikt in de *issuecrawler*¹. De *issuecrawler* is ontworpen door het *govcom.org*-team van Richard Rogers², waarvan ik deel uitmaak. Het is een webcrawler die hyperlinks van een door de gebruiker opgegeven verzameling van webpagina's volgt en analyseert om het linkgedrag tussen organisaties (websites) inzichtelijk te maken. De *issuecrawler* gaat ervan uit dat hyperlink- en discursieve kaarten een indruk van de socio-epistemische netwerken

¹Issuecrawler, <http://www.issuecrawler.net>

²Govcom.org, <http://www.govcom.org>

op het web geven. Interessant bij deze aanpak is de veronderstelling dat organisaties hun hyperlinks met zorg selecteren. Dit leidt tot de premisse dat hyperlinks (en de 'ontbrekende links') aantonen welke onderwerpen en discussiestructuren organisaties overnemen en welke ze al dan niet aanvaardbaar vinden. Ook is vastgesteld dat organisaties inhoudelijke posities innemen en andere organisaties een bepaald standpunt toedichten [3]. Zodoende kunnen de beginselen van een debat in kaart gebracht worden.

Als logisch vervolg op het in kaart brengen van debatten op het internet via hyperlinks kan ook de inhoud van teksten als conversatiestructuur in kaart gebracht worden. Met het *govcom.org*-team werd besloten de issuescraper³ te ontwikkelen. De issuescraper is een instrument om verhalen op weblogs (blogs) en nieuwssites in kaart te brengen; de beweging van verhalen door de mediaruimte wordt ermee gevolgd. De vragen die we met de software trachten te beantwoorden zijn:

- Wat is het pad van een verhaal door de blog- en nieuwsruimte?
- Welke bronnen rapporteren over een verhaal en wanneer?
- Welke invalshoeken worden er gebruikt?
- Hoe veranderen deze invalshoeken door de tijd?

Ter beantwoording van deze vragen hebben we verschillende modules ontwikkeld, die samen een vergelijkende analyse van een verhaal door de tijd heen in de nieuws- en blogruimte maken. Aan de hand van een aantal door de gebruiker ingegeven vertrekpunten en invalshoeken, in de vorm van steekwoorden, ondervraagt de issuescraper zoekmachines voor blogs en nieuws. Ook kunnen verzamelingen van bronnen gecreëerd worden, bijvoorbeeld alleen bronnen uit het Midden-Oosten, en kunnen deze bronnen ondervraagd worden naar een verhaal en een invalshoek. De inhoud van de zo verkregen artikelen wordt geanalyseerd en de resultaten daarvan worden zichtbaar gemaakt door middel van statistieken en verhaalkaarten (story maps). Deze laatste tonen de significante relaties tussen een verhaal, de invalshoeken en de bronnen ervan. De issuescraper kan aan de hand van linkanalyse ook een blogruimte ontdekken waarbinnen een verhaal zich afspeelt. Dit alles kan ook door de tijd heen bekeken worden.

Voor de ontwikkeling van deze software moesten verscheidene problemen aangepakt worden. Allereerst moesten relevante artikelen gevonden worden om de

³Issuescraper, <http://www.issuescraper.net>

verhalen te kunnen volgen. Sinds kort houden de grote zoekmachines zoals Google, Yahoo en Altavista zich bezig met het groeperen van nieuws, waardoor het gemakkelijk vindbaar is. Door een zoekopdracht in te voeren wordt een lijst met links naar artikelen verkregen. Eenzelfde aanpak wordt gebruikt voor het vinden van blogartikelen, maar dan met in blogs gespecialiseerde zoekmachines zoals Technorati. Voor de linkanalyse was door jarenlange ervaring met de issuecrawler reeds alle expertise in huis. Voor de tekstuele analyse en de ontwikkeling van de kaarten werden verscheidene bestaande modules gebruikt, maar ook nieuwe modules ontworpen. Een van de grootste problemen bleek echter de interessante, tekstuele, gegevens uit een artikel te lichten. Niet alle tekst op een webpagina is immers bruikbaar, omdat het grootste deel in de beoogde context irrelevant is: veel van deze tekst is bijvoorbeeld reclame of navigationeel van aard. Omdat het onmogelijk is om voor elke blog- en nieuwssite manueel aan te duiden waar de interessante informatie zich bevindt, ben ik op zoek gegaan naar een leeralgoritme dat automatisch de interessante informatie aanduidt.

1.2 Automatische sjablooninductie

De gemiddelde nieuws- of blogpagina bevat heel veel irrelevante informatie voor ons doel, namelijk het analyseren van de inhoud van een artikel. Voor ons doel is enkel het artikel, het eigenlijk verhaal, en niet alle navigatieve tekst, links en reclame interessant. Zoekmachines voor nieuws en blogs duiden typisch de titel, de link naar het artikel en een korte beschrijving aan. Die link kan gevolgd worden, maar levert niet enkel de inhoud van het artikel maar ook veel onzin op. Helaas duiden de meeste webpagina's niet aan welke informatie waar staat. Er moet een voorbereidingsstap gebeuren om de juiste informatie te vinden en de tekst gebruiksklaar te maken voor analyse.

Mijn scriptie behandelt precies deze vraag: hoe kan zo geautomatiseerd mogelijk het feitelijke artikel verkregen en zoveel mogelijk niet-relevante informatie weggelaten worden? Dit soort onderzoek past in een lange traditie van informatie-extractie. Bij documenten op het Wereldwijde Web wordt dit wrapperinductie genoemd. Dit wordt door Kushmerick [4] gedefinieerd als 'het aanleren van een procedure om tuples te extraheren uit een bepaalde informatiebron door middel van voorbeelden van een gebruiker'.

In machine learning zijn er twee mogelijkheden om van deze documenten te

leren: supervised en unsupervised. Bij supervised leren wordt een verzameling inputs en een bijbehorende verzameling outputs gegeven. Het leren gebeurt dan door het leeralgoritme zijn uitkomst te laten vergelijken met de vooraf aangeduide goede uitkomst zodat het weet wat zijn afwijking is en die kan aanpassen. Bij unsupervised leren wordt niet verteld wat de uitkomst is. Aan het systeem wordt overgelaten om interessante patronen, overeenkomsten of clusters te vinden.

Om een wrapper te kunnen leren moet er een zekere repetitiviteit zijn in het voorkomen of de structuur van de informatie. Elke webmaster kan de opmaak van zijn pagina inrichten zoals hij wil door middel van HTML-markeringen. Bij een grote website zal een webmaster over het algemeen dezelfde structuur en opmaak voor de verschillende pagina's van zijn site gebruiken. Dit werkt in ons voordeel, aangezien het betekent dat de site repetitiviteit kent in de structuur en opmaak van pagina's. Zo kan een wrapper geleerd worden voor elke site waaruit we nieuws of blogartikelen willen verkrijgen.

Binnen de klasse der wrappers worden twee benaderingen onderscheiden: de taggebaseerde en de structuurgebaseerde benadering. De taggebaseerde wrappers bekijken een HTML-document als een platte tekst met markeringen, de zogenaamde tags. Zulke wrappers proberen markeringseigenschappen rond de onderliggende inhoud te identificeren en gebruiken deze eigenschappen om de inhoud te extraheren. Dit werkt vooral goed als de te extraheren informatie een lineaire structuur heeft, dus als de HTML-markeringen telkens een zelfde sequentie hebben zoals in een tabel. In het domein van de nieuws- en blogartikelen hebben de gegevens over het algemeen niet zo een rigide formaat. Het doel is echter hetzelfde. HTML-documenten kunnen ook als boomstructuren bekeken worden waardoor een structuurgebaseerde wrapper geleerd kan worden die de meest specifieke veralgemening van de HTML-bomen oplevert.

In mijn scriptie heb ik een unsupervised methode gebruikt, omdat we voor de issuescraper niet voor elke weblog en nieuwssite een gelabelde voorbeeldverzameling kunnen maken. Dit zou praktisch onhaalbaar zijn. Aangezien het probleem van de unsupervised wrapperinductie nog altijd niet opgelost is, zijn er geen standaard algoritmes waarop ik me heb kunnen baseren. Reis [5] en Hogue [6] bespreken de boomwijzigingsafstand als een veelbelovende methode voor wrapperinductie. De besproken leeralgoritmes zijn structuurgebaseerd en gebruiken de boomwijzigingsafstand voor het verkrijgen van een afbeelding tussen twee HTML-bomen of delen daarvan; deze afbeelding geeft aan waar de bomen structureel gelijk zijn.

Tijdens de bestudering van deze artikelen is me opgevallen dat de auteurs een aantal zaken over het hoofd gezien hebben en dat de berekening van de optimale afbeelding erg complex is. In deze scriptie analyseer ik hoe een sjabloon gecreëerd kan worden met behulp van de boomwijzigingsafstand. Tussen een verzameling gerelateerde HTML-pagina's van eenzelfde site wordt dan de meest specifieke veralgemening van hun boomstructuur gezocht en worden de verschillen tussen de bomen door jokers gerepresenteerd. Het zo verkregen sjabloon kan dan gebruikt worden voor de extractie van de interessante gegevens, door die delen van de HTML-pagina te extraheren die gerepresenteerd worden door de jokers. Ook stel ik een algoritme voor dat een lagere gemiddelde complexiteit heeft dan de tot nu toe beschreven boomwijzigingsafstanden, door domeingeoriënteerde beperkingen aan het leeralgoritme op te leggen. Tenslotte bekijk ik hoe de geleerde sjablonen zo compact en expressief mogelijk gemaakt kunnen worden door een aantal nabewerkingsstappen toe te passen en te vergelijken.

Hoofdstuk 2

Herhaling in de structuur van HTML-bomen

Wrapperinductie is alleen mogelijk op gegevens die herhaald voorkomen. In deze scriptie wordt dan ook gekeken naar groeperingen van HTML-tags met eenzelfde structuur of, beter gezegd, de boomrepresentatie daarvan. In de volgende hoofdstukken zal worden uitgelegd dat de meest specifieke veralgemening van bomen geleerd kan worden. Doordat met nieuws- en blogartikelen in HTML gewerkt wordt is er een duidelijk domein waarbinnen mijn onderzoek zich afspeelt. In dit hoofdstuk wordt een aantal kenmerken van dit domein besproken die belangrijk zijn bij het leren.

2.1 Herhaling

De weergave van relationele gegevens op het web gebeurt meestal door middel van CGI, PHP, ASP of andere webscripts die pagina's automatisch genereren. Deze scripts halen relationele gegevens uit een gegevensbank en publiceren ze dynamisch op het web door middel van HTML-sjablonen (templates). Elk sjabloon heeft ruimte voor een bepaalde deelverzameling van de gegevens. Het script vult dit sjabloon in met de resultaten van een zoekopdracht (query) in een gegevensbank en stuurt de zo geconstrueerde HTML-pagina door naar de browser.

De sjablonen geven webpagina's hun *syntactische structuur*. Ze leveren elementen voor het formaat, de presentatie, het lettertype en andere visuele aanduidingen. De sjablonen worden gevuld met gegevens uit een relationele gegevensbank; dit wordt de *semantische structuur* genoemd. De open ruimtes of *jokers*, in de sjablonen worden

telkens gevuld met gelijkaardige semantische gegevens. Als hetzelfde semantische type gebruikt wordt zal de auteur van een pagina dezelfde sjablonen gebruiken om ze weer te geven. In deze scriptie wil ik dit proces omdraaien; ik wil de structuur en de semantiek van deze sjablonen afleiden uit een aantal voorbeeldpagina's gegenereerd en ingevuld door hetzelfde script.

Het is belangrijk te weten dat het sjabloon-proces zich op twee manieren manifesteert. Bij een blog- en nieuwsartikel wordt er één semantische klasse per pagina weergegeven. De ingevulde jokerruimtes in de pagina geven dan de eigenschappen van de semantische klasse weer. Op bijvoorbeeld de resultaatpagina van een zoekopdracht in een zoekmachine zoals Google¹ staan echter verschillende instanties van dezelfde semantische klasse op één pagina — voor elk resultaat is er een instantie.

De syntactisch en semantisch repetitieve aard van de relationele gegevens geeft een formaat voor onze sjablonen aan. Elk sjabloon moet een algemene klasse van de gegevens worden die we proberen te extraheren. Dit sjabloon moet de elementen bevatten die gemeenschappelijk zijn aan twee verschillende voorbeelden, maar moet jokers openlaten voor de voorbeeld-specifieke eigenschappen. Wanneer we een sjabloon gebruiken voor extractie, vergelijken we dit met een instantie van hetzelfde semantische type en extraheren we die elementen die aangeduid worden door de jokers in het sjabloon.

2.2 HTML-bomen

XHTML is een markeringstaal met dezelfde expressiviteit als HTML maar met een strictere syntax en een onderliggende boomstructuur die beschrijft hoe het document ingedeeld is. Met behulp van bijvoorbeeld Tidy [7] kunnen HTML-pagina's omgezet worden in XHTML zodat ze een syntactisch juiste structuur krijgen. Sinds XHTML gespecificeerd is kunnen dus alle HTML-documenten als een DOM²-boom voorgesteld worden. In deze scriptie zal telkens HTML gebruikt worden voor XHTML en HTML, opgeschoond door middel van Tidy.

Beginnende bij de oorsprong, de wortel (root) van het document, geeft elke knoop in de boom specifiekere informatie aan de browser over hoe elementen ingedeeld moeten worden. Door de manier waarop HTML geïnterpreteerd wordt worden knopen in dezelfde deelboom van het document als doorlopende blokken in de

¹Google, <http://www.google.com>

²Document Object Model

browser getoond. Belangrijk is dat deze relatie twee kanten op werkt: aangrenzende elementen in de browser worden ondersteund door knopen in dezelfde deelboom als de HTML van de pagina. Dit betekent dat een instantie van een weergegeven semantische soort op een bepaald deel van de pagina weergegeven wordt door een deelboom van knopen in de documentboom. Deze relatie kan ook opgevat worden als het resultaat van het vullen van sjablonen, zoals hiervoor beschreven. Auteurs van webpagina's neigen ertoe om gegevens structureel te groeperen. Deze groepering maakt het gemakkelijker om sjablonen 'aan elkaar te plakken' tot een volledige pagina. Evenzo maakt het hergebruik van gelijkaardige gegevens op verschillende pagina's gemakkelijker.

Behalve de algemeen herhalende structuur van semantische types merken we ook op dat maar een deel van de patronen onderhevig is aan verandering tussen de voorbeelden. In bijna alle gevallen zijn de knopen die verschillen bladeren van de boom; in sommige gevallen zijn het hele deelbomen [5]. De 'bovenste' structurele elementen, de interne knopen van de deelboom, zijn meestal dezelfde voor instanties van hetzelfde type.

Dit idee, dat de inhoud van de semantische instanties verschilt maar dat de interne structuur dezelfde is, biedt de mogelijkheid om het 'sjabloon/joker' idee te formaliseren. We gebruiken het gemeenschappelijke, interne deel van de bomen van de instanties als ons sjabloon, en we laten de instantie-specifieke bladknopen open. Deze open knopen — of jokers — kunnen dan later bij de extractie met semantische gegevens gevuld worden.

2.3 De deelboomstructuur

Zoals eerder gezegd kunnen er een of meerdere semantische instanties op één webpagina staan. Ook de structuur van een semantische instantie kan verschillende vormen aannemen. Beide worden in deze sectie verder uitgelegd.

Hogue [6] heeft de meest voorkomende structuren van de semantische gegevens op populaire websites bestudeerd. Ook hij stelde vast dat de structuur van interessante gegevens vaak herhaald wordt binnen een site. Reis [5] kwam tot een gelijkaardige conclusie; er zijn veel repetitieve gegevens omdat auteurs van webpagina's sjablonen en relationele gegevensbanken gebruiken.

Hogue deelde deze structuren in langs twee assen:

- *Overkoepelende (spanning) elementen* zijn knopen die een enkele instantie weer-

geven en gegroepeerd zijn onder één enkele knoop of verspreid over verschillende naburige knopen.

- *Semantische verwanten (siblings)*: Instanties van dezelfde klasse of eigenschap kunnen verwanten zijn die een ouderknoop delen, of ze kunnen over verschillende delen van de documentboom (of zelfs over verschillende pagina's) verspreid zijn.

Een instantie van een semantische klasse heeft meestal een structuur met één overkoepelend element. Dit is een handige structuur, aangezien de boomwijzigingsafstand waarmee de verschillen tussen de bomen gezocht zullen worden, voor gewortelde bomen gedefinieerd is. Zulke instanties kunnen verscheidene malen voorkomen op één pagina of verspreid zijn over verschillende pagina's. In andere gevallen is een semantische instantie een samenstelling van verschillende naburige deelbomen. Van dit soort structuren is het moeilijker sjablonen te maken, aangezien de boomwijzigingsafstand niet gedefinieerd is voor een disjuncte unie van bomen (ook wel bos genoemd).

Om een sjabloon te kunnen leren moeten er natuurlijk ook voorbeelden, een trainingsverzameling, zijn. De voorbeelden hangen af van de structuur van de semantische klasse. Als we gegevens hebben van het type "één gewenste semantische instantie per pagina" kan een hele webpagina als leervoorbeeld genomen worden. Dit gebeurt dus als het sjabloon van een verzameling van complete blog- of nieuwsartikelen geleerd moet worden. Als er meerdere instanties van hetzelfde type op een pagina staan, bijvoorbeeld bij zoekresultaten, wordt zo een instantie meestal gerepresenteerd als een *deelboom* van de volledige DOM-boom die de webpagina beschrijft. In andere gevallen ligt één instantie verspreid over verschillende, naast elkaar liggende, deelbomen. In beide gevallen bestaan de leervoorbeelden dus niet uit hele pagina's maar wordt dan met behulp van een door mij ontwikkelde Firefox-extensie in een webpagina aangegeven wat een instantie omvat: welke deelbomen moeten gebruikt worden om op te leren.

Samenvattend kan gesteld worden dat we een sjabloon willen leren uit een verzameling semantisch en structureel gerelateerde HTML-bomen. Door middel van de boomwijzigingsafstand, die in het volgende hoofdstuk besproken wordt, kunnen we de overeenkomsten tussen de bomen vinden. Deze overeenkomsten worden gebruikt als basis voor een HTML-sjabloon; de verschillen zullen aangeduid worden door middel van speciale jokerknopen.

Hoofdstuk 3

Wijzigingsafstanden

Nu geweten is dat HTML-pagina's een boomstructuur hebben en dat de interessante gegevens in de bladeren of als deelbomen nabij de bladeren zitten, kan gekeken worden naar methodes die bomen vergelijken. In dit hoofdstuk wordt aangetoond dat de methode die een string T omzet in een string P , de *stringwijzigingsafstand*, veralgemeend kan worden om een boom B_1 in een boom B_2 om te zetten. Door dit proces iteratief toe te passen voor meerdere bomen kan een veralgemeende boom, het sjabloon, gemaakt worden dat de bomen representeert en de verschillen in de bomen aanduidt als jokerknopen. Aan het einde van dit hoofdstuk wordt een variant van de *boomwijzigingsafstand* behandeld. In hoofdstuk 4 wordt besproken hoe een sjabloon gebruikt kan worden voor de extractie van relevante brokken informatie uit een HTML-pagina.

3.1 De wijzigingsafstand voor strings

De Levenshteinafstand of wijzigingsafstand $D(P, T)$ tussen een patroonstring P en een tekststring T is het minimaal aantal wijzigingen om T in P om te zetten. De mogelijke wijzigingen zijn de vervanging, invoeging of verwijdering van een karakter. Zo kan $T=bcdeffghixkl$ omgezet worden in $P=abcdefghijkl$ met precies drie wijzigingen:

- $bcdeffghixkl \rightarrow bcdeffghijkl$ (vervanging van x door j)
- $bcdeffghijkl \rightarrow bcdefghijkl$ (verwijdering van f)
- $bcdefghijkl \rightarrow abcdefghijkl$ (invoeging van a)

Het probleem lijkt moeilijk omdat beslist moet worden waar de invoeg- en verwijderwijzigingen in het patroon en de tekst moeten gebeuren. Door het probleem

te herformuleren als: ‘Wat moet er gebeuren met elk laatste karakter in een string?’ wordt het al wat gemakkelijker. De laatste karakters kunnen gelijk zijn of moeten vervangen worden. De enige andere mogelijkheid voor de wijziging van het laatste karakter van het patroon of de tekst zijn de invoeging of verwijdering van een karakter. De wijzigingsafstand kan dan als volgt gedefinieerd worden:

Definitie 1 (Wijzigingsafstand) *Laat $D[i][j]$ het minimale aantal wijzigingen tussen $P = P_1P_2P_3 \dots P_i$ en het deel van T dat eindigt in j zijn. Dan is $D[i][j]$ het minimum van de drie mogelijke manieren om de ene string in de andere om te zetten:*

1. $D[i - 1, j] + \text{invoegprijs}(P_i)$ (er moet een extra karakter ingevoegd worden),
2. $D[i, j - 1] + \text{verwijderprijs}(T_j)$ (er moet een extra karakter verwijderd worden),
3. Als $(P_i \neq T_j)$ dan $D[i - 1][j - 1] + \text{vervangingsprijs}(P_i, T_j)$ en anders $D[i - 1][j - 1]$ (de i -de en de j -de karakters moeten vervangen worden of zijn gelijk).

Door de vaststelling dat de prijs van een wijziging telkens van de prijs van de vorige wijziging afhangt, is het simpel een dynamisch programmeeralgoritme te schrijven, dat een $m + 1$ bij $n + 1$ matrix D creëert met $m = |P|$ en $n = |T|$. De pseudocode hiervan staat in Figuur 3.1. Ter illustratie staat de prijsmatrix die geproduceerd wordt bij het omzetten van $T=bcdeffghixkl$ in $P=abcdefghijkl$ in Figuur 3.1.

Voor de grenscondities geldt dat $D[0][0] = 0$. De prijs van de verwijdering van de eerste i karakters is $D[0][i] = i$, dat wil zeggen de prijs van de omzetting van de eerste i karakters van de tekst in geen enkel karakter van het patroon. De initialisatie van de prijs voor de invoegingen gebeurt op analoge wijze. Als bij vergelijking twee karakters gelijk zijn is de prijs 0.

Als alleen de optimale prijs voor de wijziging van T in P gewenst is kan gewoon de prijs van $D[m][n]$ uitgelezen worden. Deze optimale prijs drukt dan de kleinste afstand, het minimaal aantal wijzigingen om T in P om te zetten, uit. De benodigde tijd om deze prijs te berekenen is $O(m \cdot n)$, omdat er voor het invullen van cel $D[i][j]$ maar twee karakters vergeleken worden, er naar drie cellen gekeken wordt en de tijd voor het invullen van een cel constant is.

Als de optimale wijzigingsprijs gekend is kan ook het optimale wijzigingsscript gevonden worden. Een wijzigingsscript S tussen twee strings T en P is een opeenvolging van wijzigingen die T in P omzet. De prijs van S , $\gamma(S)$, is de som van de prijzen van de wijzigingen in S . Een optimaal wijzigingsscript tussen T en P is het script

```

1 Wijzigingsafstand( P, T )
2   /* initialisatie */
3   m = grootte( P )
4   n = grootte( T )
4   for i = 0 to m do D[i][0] = i; end
5   for j = 0 to n do D[0][j] = j; end
6   /* herhaling */
7   for i = 1 to m do
8     for j = 1 to n do
9       D[i][j] = min( D[i][j-1] + verwijderprijs(T[j]),
10                    D[i-1][j] + invoegprijs(P[i]),
11                    D[i-1][j-1] + vervangingsprijs(P[i],T[j])
12                    )
13     end
14   end
15   return D[m][n]
16 end

```

Figuur 3.1: De pseudocode voor de berekening van de wijzigingsafstand door middel van dynamisch programmeren.

		b	c	d	e	f	f	g	h	i	x	k	l
	0	1	2	3	4	5	6	7	8	9	10	11	12
a	1	<u>1</u>	2	3	4	5	6	7	8	9	10	11	12
b	2	<u>1</u>	2	3	4	5	6	7	8	9	10	11	12
c	3	2	<u>1</u>	2	3	4	5	6	7	8	9	10	11
d	4	3	2	<u>1</u>	2	3	4	5	6	7	8	9	10
e	5	4	3	2	<u>1</u>	2	3	4	5	6	7	8	9
f	6	5	4	3	2	<u>1</u>	<u>2</u>	3	4	5	6	7	8
g	7	6	5	4	3	2	2	<u>2</u>	3	4	5	6	7
h	8	7	6	5	4	3	3	3	<u>2</u>	3	4	5	6
i	9	8	7	6	5	4	4	4	3	<u>2</u>	3	4	5
j	10	9	8	7	6	5	5	5	4	3	<u>3</u>	4	5
k	11	10	9	8	7	6	6	6	5	4	4	<u>3</u>	4
l	12	11	10	9	8	7	7	7	6	5	5	4	<u>3</u>

Figuur 3.2: Voorbeeld van een matrix gecreëerd voor de berekening van de wijzigingsafstand tussen van $T=bcdeffghixkl$ en $P=abcdefghijkl$. De prijs van elke wijziging is hier 1. De optimale afbeelding is onderstreept.

met de laagste prijs. Om S te bepalen kan de prijismatrix achterwaarts afgelopen worden vanuit de cel met de laagste prijs voor de omzetting van T in P , $D[m][n]$. Om in cel $D[m][n]$ aan te komen is een van de cellen $D[n-1][m]$ (invoeging), $D[n, m-1]$ (verwijdering) of $D[n-1][m-1]$ (verandering of gelijkheid) gebruikt. Welke van de drie mogelijkheden gekozen wordt hangt af van de prijs die weergegeven is in deze cellen en de karakters $P[n]$ en $T[m]$. Door dit proces te herhalen voor elke voorgaande cel kan het hele pad door de matrix — en dus S — gereconstrueerd worden. S vertelt dus welke karakters zijn vervangen, verwijderd of ingevoegd en op welke plaats dit is gebeurd. De achterwaartse gang door de matrix heeft als complexiteit $O(n+m)$, aangezien alleen die cellen afgelopen moeten worden die in de uitlijning van toepassing zijn.

3.2 De boomwijzigingsafstand

Naar analogie met de wijzigingsafstand voor strings is de boomwijzigingsafstand tussen twee bomen B_1 en B_2 gedefinieerd als de prijs van een opeenvolging van wijzigingen waarbij B_1 in B_2 omgezet wordt. In plaats van met karakters zal hier met knopen en hun kinderen gewerkt worden. Eerst komen de notatie en wat preliminaire definities aan bod, waarna de veralgemening van strings naar bomen getoond wordt. De complexiteit van deze algoritmes wordt besproken en in hoofdstuk 5 wordt mijn eigen verbetering besproken.

3.2.1 Definities

Voor een graaf G beschrijven we de knopen en zijden respectievelijk als $V(G)$ en $E(G)$. B is een gewortelde boom waarvan de wortel, of oorsprong, aangeduid wordt met $\text{oorsprong}(B)$. De *grootte* van B , beschreven door $|B|$, is $|V(B)|$. De *absolute diepte* van een knoop $v \in V(B)$, $\text{diepte}_{abs}(v)$, is het aantal zijden op een pad van v naar $\text{oorsprong}(B)$. De *relatieve diepte*, $\text{diepte}_{rel}(v_1, v_2)$ tussen twee knopen v_1 en v_2 is het aantal zijden op een pad van v_1 naar v_2 . Een knoop w heeft een *ouder* v als er een knoop met een relatieve diepte van 1 boven hem in de boom zit en wordt aangeduid als $\text{ouder}(w) = v$. Omgekeerd is w een *kind* van v , $\text{kind}(v) = w$. De *in-graad* (in-degree) van een knoop v , $\text{graad}(v)$, is het aantal kinderen van v . Knopen met een relatieve diepte ≥ 1 en lager in de boom dan v worden *afstammelingen* genoemd. De *voorouders* van v zijn alle knopen op het pad van v naar de oorsprong. We breiden

deze definities zo uit dat $\text{diepte}(B)$ en $\text{graad}(B)$ respectievelijk de maximale absolute diepte en het maximaal aantal kinderen van een willekeurige knoop in B beschrijven. Een knoop zonder kinderen is een *blad* en anders een interne knoop. Het aantal bladeren van B wordt aangeduid met $\text{bladeren}(B)$. Twee knopen zijn *verwant* als ze dezelfde ouder hebben. Voor twee bomen B_1 en B_2 worden $\text{bladeren}(B_i)$, $\text{diepte}(B_i)$ en $\text{graad}(B_i)$ respectievelijk beschreven als L_i , D_i en G_i met $i = 1, 2$. Een lege boom wordt weergegeven door λ en $B(v)$ is de deelboom van B geworteld in $v \in V(B)$. Een *niveau* omvat alle knopen op een bepaalde absolute diepte. Een *deelniveau* van een knoop v , $\text{deelniveau}(v)$, omvat alle kinderen van v .

Een boom B is *geordend* als er een links-naar-rechts ordening tussen de verwanten bestaat. Voor een geordende boom B met oorsprong v en kinderen v_1, \dots, v_i , wordt de *pre-orde doorkruising* (preorder traversal) van $B(v)$ verkregen door eerst v en dan recursief de knopen $B(v_k)$, $1 \leq k \leq i$ te bezoeken. Op gelijke wijze wordt de *postorde doorkruising* verkregen door eerst $B(v_k)$, $1 \leq k \leq i$ en dan v te bezoeken. Het *pre-orde getal* en het *postorde getal* van een knoop $w \in B(v)$, aangeduid met $\text{pre}(w)$ en $\text{post}(w)$, is het aantal knopen voor w in de pre-orde en postorde doorkruising van B . De knopen *links* van w in B is de verzameling knopen $u \in V(B)$, zodat $\text{pre}(u) < \text{pre}(w)$ en $\text{post}(u) < \text{post}(w)$. Als u links ligt van w dan ligt w rechts van u .

Een bos is een verzameling van bomen. Een bos F is geordend als er een links-naar-rechts ordening bestaat tussen de bomen en elke boom geordend is. Als B een geordende boom is en $v \in V(B)$ en v_1, \dots, v_i de kinderen zijn van v , kan $F(v_s, v_t)$ met $1 \leq s \leq t \leq i$ gedefinieerd worden als het bos $B(v_s), \dots, B(v_t)$. Met $F(v)$ wordt $F(v_1, v_i)$ aangeduid.

Een boom is *gelabeld* als alle knopen een label ℓ uit een eindig alfabet Σ hebben. Het label van een knoop v wordt aangeduid als $\text{lab}(v)$. Als $\lambda \notin \Sigma$ een speciale *lege knoop* is dan definiëren we Σ_λ als $\Sigma \cup \{\lambda\}$. Tot slot definiëren we een *prijfsfunctie* op labelparen, $\gamma: (\Sigma_\lambda \times \Sigma_\lambda) \setminus \{(\lambda, \lambda)\} \rightarrow \mathbb{R}$. γ is een afstandsmaat, wat wil zeggen dat voor elke ℓ_1, ℓ_2, ℓ_3 de volgende condities gelden:

1. $\gamma(\ell_1, \ell_2) \geq 0, \gamma(\ell_1, \ell_1) = 0,$
2. $\gamma(\ell_1, \ell_2) = \gamma(\ell_2, \ell_1),$
3. $\gamma(\ell_1, \ell_3) \leq \gamma(\ell_1, \ell_2) + \gamma(\ell_2, \ell_3).$

3.2.2 Wijzigingen en afbeeldingen

Als B_1 en B_2 twee gelabelde en geordende bomen zijn, wordt elke wijziging weergegeven door $(\ell_1 \rightarrow \ell_2)$ met $(\ell_1, \ell_2) \in (\Sigma_\lambda \times \Sigma_\lambda) \setminus \{(\lambda, \lambda)\}$. De wijziging is een vervanging als $\ell_1 \neq \ell_2$, een verwijdering als $\ell_2 = \lambda$ en een invoeging als $\ell_1 = \lambda$. De notatie wordt zo uitgebreid dat $(v \rightarrow w)$, voor knopen v en w , $(\text{lab}(v) \rightarrow \text{lab}(w))$ betekent. Net zoals bij labels kunnen v of w λ zijn. Een vervanging van knoop v betekent de wijziging van het label. Een knoop v verwijderen gebeurt door van de kinderen van v de kinderen van de ouder van v te maken en v te verwijderen. De invoeging van een knoop is het complement van de verwijdering van een knoop. Dit betekent dat als een knoop v ingevoegd wordt als het kind van w , v de ouder wordt van een subsequentie van de huidige kinderen van w .

Gegeven de prijsfunctie γ op paren van labels kan de prijs van een wijziging genoteerd worden als $\gamma(\ell_1 \rightarrow \ell_2) = \gamma(\ell_1, \ell_2)$. De prijs van een opeenvolging $S = s_1, \dots, s_k$ wijzigingen is gegeven door $\gamma(S) = \sum_{i=1}^k \gamma(s_i)$. De *boomwijzigingsafstand*, $\delta(B_1, B_2)$ wordt dan formeel gedefinieerd als:

Definitie 2 (Boomwijzigingsafstand)

$$\delta(B_1, B_2) = \min\{\gamma(S) \mid S \text{ is een opeenvolging van wijzigingen die } B_1 \text{ in } B_2 \text{ omzet}\}$$

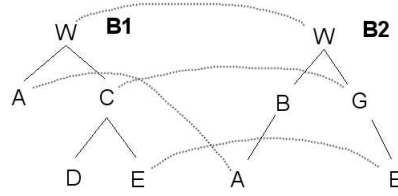
Aangezien γ een afstandsmaat is wordt ook δ een afstandsmaat.

Een *afbeelding* (mapping) tussen B_1 en B_2 is een grafische weergave van de wijzigingen. Formeel wordt dit als volgt gedefinieerd:

Definitie 3 (Afbeelding) *Het triple (M, B_1, B_2) is een geordende afbeelding tussen B_1 en B_2 als voor $M \subseteq V(B_1) \times V(B_2)$ en voor elk paar $(v_1, w_1), (v_2, w_2) \in M$ het volgende geldt:*

1. $v_1 = v_2$ als en slechts als $w_1 = w_2$ (één-op-één relatie),
2. v_1 is een voorvader van v_2 als en slechts als w_1 een voorvader is van w_2 (voorvaderrelaties blijven behouden),
3. v_1 zit links van v_2 als en slechts als w_1 links zit van w_2 (de volgorde van de verwanten blijft ongewijzigd).

In plaats van (M, B_1, B_2) wordt M gebruikt als er geen verwarring mogelijk is. Figuur 3.3 illustreert een dergelijke afbeelding. Als (M, B_1, B_2) een afbeelding is zeggen we dat een knoop v in B_1 of B_2 *uitgelijnd* wordt door een lijn in M als v voorkomt in een paar dat in M zit. Als N_1 en N_2 de verzamelingen knopen in respectievelijk B_1 en B_2 zijn die niet in de mapping voorkomen, dan wordt de prijs van M gegeven door:



Figuur 3.3: Voorbeeld van een afbeelding waarbij de wijzigingen voor de omzetting van B_1 in B_2 de invoeging van B, de verwijdering van D en de vervanging van C door G zijn.

Definitie 4 (De prijs van een afbeelding)

$$\gamma(M) = \sum_{(v,w) \in M} \gamma(v \rightarrow w) + \sum_{v \in N_1} \gamma(v \rightarrow \lambda) + \sum_{w \in N_2} \gamma(\lambda \rightarrow w),$$

Afbeeldingen kunnen ook samengesteld worden. Als B_1, B_2 en B_3 bomen zijn en M_1 en M_2 respectievelijk afbeeldingen van B_1 naar B_2 en van B_2 naar B_3 , dan geldt de volgende definitie:

Definitie 5 (Samengestelde afbeeldingen)

$$M_1 \circ M_2 = \{(v, w) \mid \exists u \in V(B_2) \text{ zodat } (v, u) \in M_1 \text{ en } (u, w) \in M_2\}$$

Het is duidelijk dat $M_1 \circ M_2$ zelf een afbeelding wordt van B_1 naar B_3 . Omdat γ een maat is kan ook gemakkelijk aangetoond worden dat een afbeelding met een minimale prijs equivalent is aan de wijzigingsafstand:

Definitie 6 (De equivalentie van $\delta(M)$ en $\delta(B_1, B_2)$)

$$\delta(B_1, B_2) = \min\{\delta(M) \mid (M, B_1, B_2) \text{ is een afbeelding voor de wijzigingsafstand}\}.$$

Door de berekening van de wijzigingsafstand kan dus de optimale afbeelding berekend worden. De prijs van een individuele wijziging wordt als volgt gedefinieerd:

Definitie 7 (Prijs van een individuele wijziging)

$$\gamma(v \rightarrow w) = \begin{cases} 1 & \text{als } v \neq w \\ 0 & \text{als } v = w \end{cases}$$

Let wel: de vergelijking van een knoop met λ faalt.

Net zoals bij de berekening van de minimale wijzigingsafstand voor strings kan voor de berekening van de minimale afbeelding tussen B_1 en B_2 gebruik gemaakt

worden van een dynamische programmeerstrategie. Omdat het hier over bomen gaat moet echter ook geprobeerd worden de deelbomen, die v en w als oorsprong hebben, met elkaar af te lijnen. Daarvoor zal recursief de optimale afbeelding tussen de kinderknopen $v[i]$ en $w[j]$ van v en w gezocht worden. Stel $m = \text{graad}(v)$ en $n = \text{graad}(w)$. We creëren dan een $m + 1$ bij $n + 1$ matrix D . $D[i][j]$ bevat de afbeelding M_{ij} tussen $v[i]$ en $w[j]$, voor $0 \leq i \leq m + 1$ en $0 \leq j \leq n + 1$. We kunnen dan de cellen van D berekenen door vast te stellen dat:

$$D[i][j] = \min_{\gamma} \begin{cases} D[i-1][j] & + \text{invoegprijs}(w[j]) \\ D[i][j-1] & + \text{verwijderprijs}(v[i]) \\ D[i-1][j-1] & + \text{optimale-afbeelding}(v[i], w[j]). \end{cases}$$

$D[0][0]$ is gedefinieerd als een lege afbeelding. $D[m][0]$ en $D[0][n]$ stellen afbeeldingen voor waar respectievelijk alle knopen van v verwijderd en alle knopen van w ingevoegd zijn. De *optimale-afbeelding*($v[i], w[j]$) wordt recursief berekend.

Tijdens de berekening van de optimale afbeelding slaan we voor elke m en n de wijzigingen op, evenals de afstammingspaden van de betreffende knopen, om zo een wijzigingsscript S te maken. Als de volledige matrix D berekend is kunnen we deze heel simpel uitlezen en $D[m][n]$ als de optimale afbeelding tussen v en w weergeven, met het bijbehorende optimale wijzigingsscript S .

3.2.3 Complexiteit

De berekening van de boomwijzigingsafstand van een geordende boom is gecompliceerd; verschillende algoritmes met verschillende afwegingen zijn voorgesteld maar alle hebben ze een groter dan kwadratische complexiteit [8]. Ook is bewezen dat als de bomen niet geordend zijn, het probleem NP-compleet is [9]. Het eerste algoritme voor het afbeeldingsprobleem is beschreven in [10] en heeft een complexiteit van $O(|B_1| \cdot |B_2| \cdot D_1^2 \cdot D_2^2)$. Dit is een dynamisch programmeeralgoritme dat recursief de wijzigingsafstand berekent tussen de strings gevormd door de verzamelingen van de kinderknopen van elke interne knoop van de boom. De beste bovengrens van de prijs wordt voorgesteld in [8] met een complexiteit van $O(|B_1| \cdot |B_2| + L_1^2 \cdot |B_2| + L_1^{2.5} \cdot L_2)$.

3.3 Beperkte top-down boomwijzigingsafstand

Ondanks de grote complexiteit van het afbeeldingsprobleem zijn er toch verschillende praktische toepassingen door extra voorwaarden of beperkingen aan de wij-

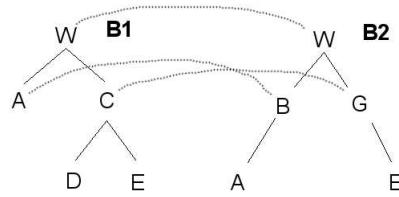
zizingen van definitie 3 toe te voegen [11].

Binnen het domein van een verzameling automatisch gegenereerde HTML-bomen van eenzelfde site komen de verschillen tussen de pagina's in of nabij de bladeren voor, in sommige gevallen zijn het hele deelbomen. Dit betekent dat het bovenste, grootste, deel van de bomen hetzelfde is in alle pagina's. Ook geldt binnen dit domein dat aangrenzende elementen in een pagina gerepresenteerd worden in dezelfde deelboom dicht bij de bladeren. Deze observaties kunnen in de berekening van de boomwijzigingsafstand opgenomen worden door alleen nog maar elementen van hetzelfde niveau in de te vergelijken bomen te bekijken en de wijzigingen te beperken tot de bladeren. Door de wijzigingen tot de bladeren te beperken en deze wijzigingen later te representeren door jokerknopen kan dan in principe een sjabloon gemaakt worden dat de verschillen tussen een verzameling van HTML-pagina's aanduidt. Het is de bedoeling dat het sjabloon de meest specifieke veralgemening van een verzameling bomen wordt, waarbij de jokers aanduiden wat pagina-specifiek is. Op deze manier zijn jokers zinvolle representaties van unieke elementen in een HTML-pagina. Dit wordt verder besproken in het volgende hoofdstuk maar eerst zal bekeken worden hoe de definitie van de boomwijzigingsafstand aangepast moet worden om te zorgen dat hij top-down wordt en de wijzigingen beperkt worden tot de bladeren.

Definitie 8 (Beperkte top-down afbeelding) *Het triple (M, B_1, B_2) is een beperkt top-down afbeelding tussen B_1 en B_2 als voor $M \subseteq V(B_1) \times V(B_2)$ en voor elk paar $(v_1, w_1), (v_2, w_2) \in M$ het volgende geldt:*

1. $v_1 = v_2$ als en slechts als $w_1 = w_2$ (één-op-één relatie),
2. $(ouder(v_1), ouder(w_1)) \in M$ waar v_1 en w_1 geen oorsprongsknopen van B_1 of B_2 zijn (als een paar knopen in de afbeelding zit zitten ook hun ouders in de afbeelding),
3. v_1 zit links van v_2 als en slechts als w_1 links zit van w_2 (de volgorde van de verwanten blijft ongewijzigd),
4. als v_1 de voorouder is van v_2 dan moet ook $label(v_1) = label(w_1)$ (de voorouders moeten hetzelfde zijn hebben).

Het eerste en derde deel van definitie 8 zijn ongewijzigd ten op zichte van definitie 3. Deel twee, dat eerst stelde dat als een knoop een voorouder is ook de daarmee uitgelijnde knoop een voorouder moet zijn, is nu strenger geworden en zegt dat



Figuur 3.4: Voorbeeld van een beperkte top-down afbeelding.

alle voorouders van een knoop ook in de afbeelding moeten zitten. Dit maakt de afbeelding top-down. Deel 4 van deze definitie stelt dat al die voorouders ook hetzelfde label moeten hebben. Dit heeft tot gevolg dat als een knoop wijzigt, geen van zijn afstammelingen in de afbeelding voorkomt. Het is gemakkelijk in te zien dat als v afgebeeld wordt op w ook $\text{ouder}(v)$ op $\text{ouder}(w)$ afgebeeld moet worden. De reden is dat als v verwijderd is zijn ouder niet verwijderd, ingevoegd of vervangen kan worden. Dit betekent dat als twee knopen uitgelijnd worden ook hun ouders uitgelijnd moeten worden. Figuur 3.4 is een voorbeeld van een beperkte top-down afbeelding.

Merk op dat door de tweede en de derde voorwaarde in definitie 4 de invoeging of verwijdering van een knoop evenveel kost als de invoeging of verwijdering van de hele deelboom met die knoop als oorsprong. Als de labels van knoop v en knoop w niet gelijk zijn is de prijs van de vervanging de prijs van het vervangen van de knoop plus de prijs van het verwijderen van $F_1(v)$ en het invoegen van $F_2(w)$. Als de labels hetzelfde zijn wordt er gerecurseerd om de optimale afbeelding van $F(v)$ en $F(w)$ te berekenen.

Ook hier kan de afstand tussen twee bomen B_1 en B_2 gedefinieerd worden als de minimale prijs van een beperkte top-down afbeelding tussen twee bomen, al zal deze niet dezelfde zijn als de prijs van de gewone boomwijzigingsafstand. Dit komt omdat deelbomen vaak als één entiteit bekeken worden — doordat hele deelbomen in een keer verwijderd of ingevoegd kunnen worden.

Aangezien in ons domein de veranderingen zich in of nabij de bladeren bevinden en we willen dat de bomen structureel zo veel mogelijk gelijk blijven, is de beperkte top-down boomwijzigingsafstand een goede manier om dit te verzekeren. De resulterende afbeelding geeft aan waar de bomen gelijk zijn of van label moeten veranderen. Knopen die niet in de afbeelding zitten moeten verwijderd of ingevoegd worden voor het omzetten van de ene boom in de andere. Reis probeerde dit uit

te leggen in [5] maar was niet strikt genoeg in zijn formulering en vergat de vierde voorwaarde, waardoor voor ons domein, onzinnige afbeeldingen verkregen kunnen worden. Behalve de net beschreven top-down shortcut maakte hij in zijn implementatie ook een bottom-up shortcut bij de berekening van de wijzigingsafstand: van tevoren wordt berekend welke bomen identiek zijn. In het slechtste geval heeft Reis' implementatie van de boomwijzigingsafstand een prijs $O(|B_1| \cdot |B_2|)$, maar dit komt enkel voor als de bomen, behalve in hun bladeren, identiek zijn. In de praktijk wordt de top-down shortcut, die enkel wijzigingen in de blaadjes toelaat waardoor hele deelbomen niet vergeleken hoeven te worden, vaak gebruikt wat de prijs aanzienlijk drukt. D wordt dus alleen berekend als twee knopen hetzelfde label hebben.

3.4 Samenvatting

In dit hoofdstuk is aangetoond dat de wijzigingsafstand voor strings veralgemeend kan worden, zodat ook de verschillen tussen bomen berekend kunnen worden. Door domeinspecifieke eigenschappen op te nemen in de definitie van een afbeelding kunnen snellere algoritmes gebruikt worden voor bepaalde toepassingen. Voor het domein van de automatisch gegenereerde HTML-bomen proberen we de structuur van de bomen top-down zo veel mogelijk gelijk te houden en kunnen we de wijzigingen tot de bladeren beperken. We willen in dit domein juist weten vanaf welke knopen de bomen van elkaar verschillen, aangezien op die plaatsen de voor die pagina unieke en dus interessante informatie zit. Als twee knopen verschillen doen de onderliggende knopen er dus niet meer toe bij de berekening van de boomwijzigingsafstand. Dit leidt er toe dat het aantal berekeningen van de boomwijzigingsafstand achterwege gelaten kunnen worden voor het bepalen van de voor ons domein gewenste wijzigingsafstand. De knopen die verschillen tussen de bomen worden later gemerkt als jokerknopen. In de volgende hoofdstukken neem ik onder de loep wat er moet gebeuren om met behulp van de wijzigingsafstand sjablonen te maken en tracht ik de berekening van een optimale afbeelding te versnellen.

Hoofdstuk 4

De veralgemening van HTML-pagina's

Voor de wrapperinductie hebben we een sjabloon nodig dat de overeenkomsten tussen alle webpagina's behoudt maar de verschillen aanduidt. In het vorige hoofdstuk is uitgelegd dat de beperkte top-down afbeelding aanduidt waar de boomrepresentaties van HTML-pagina's van elkaar verschillen. Met het bijbehorende wijzigings-script kunnen de verschillen tussen de bomen dan als jokers aangeduid worden. In dit hoofdstuk wordt besproken hoe Reis [5] en Hogue [6] de sjablonen construeren. Ook wordt uitgelegd dat als er eenmaal een veralgemening tussen twee bomen uit de trainingsverzameling is gevonden, de jokerknoten uit dat sjabloon een speciale status moeten krijgen bij het zoeken van de afbeelding tussen dat sjabloon en een nieuw voorbeeld. Dit kan door de prijsfunctie van definitie 7 aan te passen. Aan het eind van dit hoofdstuk wordt uitgelegd hoe het sjabloon gebruikt kan worden voor extractie.

4.1 Sjablonen

Met behulp van de wijzigingsafstand en het bijbehorende wijzigings-script kunnen twee voorbeelden veralgemeend worden tot een sjabloon.

Definitie 9 (Sjabloon) *Een sjabloon is een geordende en gelabelde boom die speciale jokerknoten kan bevatten. Elke joker moet een blad van de boom zijn en kan een van de volgende types zijn:*

- $\text{single}(\ell)$ (.). *Deze jokerknoop moet één deelboom v uitlijnen met $\text{lab}(v) = \ell$.*
- $\text{plus}(\ell)$ (+). *Deze jokerknoop lijnt uit met een bos $F(v_i \cdots v_k)$ waar het aantal bomen in het bos ≥ 1 moet zijn en zolang $\text{lab}(v_i) = \ell$.*

- $\text{optional}(\ell)$ (?). Deze jokerknoop mag één deelboom v uitlijnen met $\text{lab}(v) = \ell$, maar hoeft dit niet te doen.
- $\text{kleene}(\ell)$ (*). Deze jokerknoop lijnt uit met een bos $F(v_i \cdots v_k)$ waar het aantal bomen in het bos ≥ 0 kan zijn en zolang $\text{lab}(v_i) = \ell$.

De uitlijning tussen een sjabloon en een boom wordt als volgt gedefinieerd:

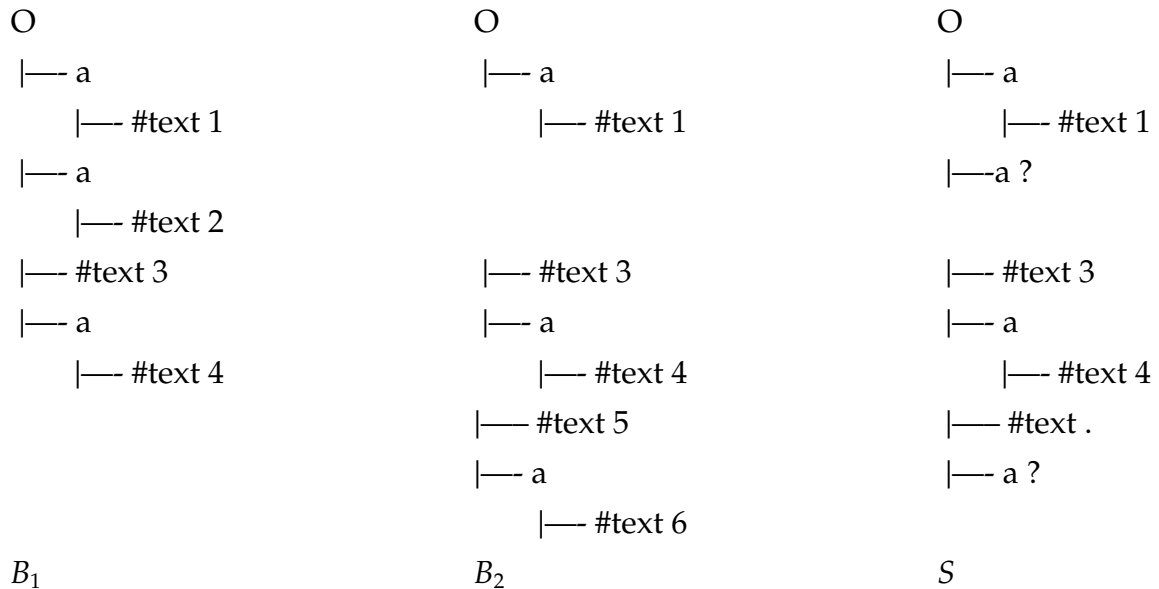
Definitie 10 (Uitlijning voor sjablonen) De uitlijning tussen een sjabloon en een doelboom is een afbeelding waarbij aan de volgende regels in volgorde voldaan wordt:

1. Elke niet-jokerknoop in het sjabloon moet afgebeeld worden op een identieke knoop in de doelboom.
2. Elke knoop in de doelboom moet afgebeeld worden op een identieke niet-jokerknoop in het sjabloon of geconsumeerd worden door een joker.
3. Single jokers (.) moeten één deelboom van de doelboom consumeren.
4. Plus jokers (+) moeten ten minste één deelboom van de doelboom consumeren.
5. Optional jokers (?) moeten indien mogelijk één deelboom van de doelboom consumeren.
6. Kleene jokers (*) moeten indien mogelijk ten minste één deelboom van de doelboom consumeren.

Anders gesteld bestaat een uitlijning tussen twee bomen als er een afbeelding zonder wijzigingen tussen die twee bomen bestaat. Merk op dat de definitie voor de uitlijning met sjablonen strenger is dan de uitlijning van een afbeelding voor het berekenen van de verschillen. De uitlijning van sjablonen staat immers geen labelvervangingen toe. Merk ook op dat een boom geïnterpreteerd kan worden als een sjabloon zonder jokers. In Figuur 4.1 staat een voorbeeld van zo een sjabloon S dat de veralgemening is van de bomen B_1 en B_2 .

4.1.1 Jokers

Een joker wordt in een sjabloon gebruikt om aan te duiden dat er bij de afbeelding een invoeging, verwijdering of vervanging heeft plaatsgevonden. In mijn implementatie wordt een knoop als jokerknoop gemerkt door hem met een speciaal HTML-attribuut 'joker' aan te duiden. Zodoende blijft de informatie over de plaats en het label van de knoop bewaard. Alle afstammelingen van een jokerknoop worden gesnoeid; een



Figuur 4.1: Een voorbeeld van het sjabloon S ; de veralgemening van de twee bomen B_1 en B_2 .

joker geeft immers aan dat alle informatie vervat in deze knoop en zijn afstammelingen verschilt in tenminste twee van de bomen uit de trainingsverzameling. Eerder is al vermeld dat we juist geïnteresseerd zijn in die dynamische inhoud: de specifieke instanties van een semantisch klasse die gerepresenteerd wordt door het sjabloon.

In het sjabloon kunnen verschillende soorten jokers gebruikt worden, elk met hun eigen interpretatie, of er kan telkens dezelfde soort jokers gebruikt worden. Hogue [6] gebruikt maar één soort jokers. In mijn implementatie heb ik dit toegepast door alle jokers uit definitie 9 als optionals te interpreteren. Reis [5] gebruikt vier soorten jokers, vergelijkbaar met die uit reguliere expressies: *single*, *plus*, *optional* en *kleene*. Een knoop wordt als *single* gemerkt in het sjabloon als er een vervanging in het wijzigingsscript staat voor die knoop. Het is de bedoeling dat deze joker overeenkomt met gewenste objecten zoals de titel van nieuws. Een knoop wordt als *optional* gemerkt voor die knopen die niet in de afbeelding zitten: die knopen die verwijderd moeten worden uit de eerste boom en ingevoegd uit de tweede boom. Het is de bedoeling dat deze joker overeenkomt met facultatieve elementen zoals lijsten met verwant nieuws. Als verschillende datarijke objecten in webpagina's zich over verschillende verwante deelbomen uitstrekken, zoals bij de tekst in een nieuwsartikel waar de inhoud uit verschillende paragrafen bestaat, kunnen al deze objecten gerepresenteerd worden met één enkele joker: een *plus* of een *kleene*. Naar

analogie met reguliere expressies moeten een single en een plus met minstens één deelboom uitlijnen en kunnen de facultatieve elementen optional en kleene met nul of meerdere deelbomen uitlijnen. De plus en de kleene worden in een nabeweringsstap geconstrueerd, nadat het sjabloon is opgebouwd. Ook Hogue vermeldt een nabeweringsstap. Beide worden besproken in hoofdstuk 4.3.

4.1.2 Het bouwen van sjablonen

Om een sjabloon te bouwen itereren we over alle bomen uit een verzameling van gerelateerde HTML-pagina's en stellen we incrementeel alle bomen samen. Merk op dat elke boom gezien kan worden als een sjabloon zonder jokers. De bedoeling is dat dit proces uiteindelijk de meest specifieke veralgemening van de bomen in de trainingsverzameling oplevert.

De beperkte top-down boomwijzigingsafstand wordt als volgt gebruikt worden om een sjabloon te bouwen. De knopen v en w van een sjabloon zijn gelijk als en slechts als:

- v en w beide jokerknopen van hetzelfde type zijn en ze dezelfde labels hebben;
- v en w geen jokers zijn en de labels hetzelfde zijn;

Gegeven twee sjablonen S_1 en S_2 wordt met de beperkte top-down boomwijzigingsafstand een afbeelding ($S_1 \rightarrow S_2$) verkregen. Met deze afbeelding kan een samengesteld sjabloon $S_3 = S_1 \circ S_2$ gebouwd worden met behulp van de volgende regels:

- $f(v, w)$ is als volgt gedefinieerd:

$$\begin{array}{l}
 f(*, *) = * \quad \left| \quad \begin{array}{l} f(+, +) = + \\ f(+, \cdot) = + \\ f(+, ?) = * \end{array} \quad \left| \quad \begin{array}{l} f(\cdot, \cdot) = \cdot \\ f(\cdot, ?) = ? \\ f(\cdot, n) = \cdot \end{array} \\
 f(*, +) = * \quad \left| \quad \begin{array}{l} f(+, \cdot) = + \\ f(+, n) = + \end{array} \quad \left| \quad \begin{array}{l} f(\cdot, ?) = ? \\ f(?, ?) = ? \end{array} \\
 f(*, ?) = * \quad \left| \quad \begin{array}{l} f(+, ?) = * \\ f(+, n) = + \end{array} \quad \left| \quad \begin{array}{l} f(\cdot, n) = \cdot \\ f(?, ?) = ? \end{array} \\
 f(*, \cdot) = * \quad \left| \quad \begin{array}{l} f(+, n) = + \end{array} \quad \left| \quad \begin{array}{l} f(?, ?) = ? \\ f(?, n) = ? \end{array} \\
 f(*, n) = * \quad \left| \quad \begin{array}{l} \end{array} \quad \left| \quad \begin{array}{l} f(?, n) = ? \end{array}
 \end{array}$$

$$f(n_1, n_2) = n_1 \text{ als } n_1 \text{ en } n_2 \text{ hetzelfde label hebben}$$

$$f(n_1, n_2) = \cdot \text{ als } n_1 \text{ en } n_2 \text{ verschillende labels hebben}$$

- als v niet in de afbeelding zit dan wordt v' aan S_3 toegevoegd met $v' = f(v, ?)$;
- als v afgebeeld wordt op w dan wordt v' aan S_3 toegevoegd met $v' = f(v, w)$;

waarbij n , n_1 en n_2 geen jokerknopen zijn. De volgorde van de parameters is niet belangrijk. Merk op dat een afbeelding alleen die knopenparen bevat waarvan de knopen ofwel gelijk zijn of vervangen moeten worden. Het tweede punt van voorgaande definitie zorgt er voor dat knopen die niet in de afbeelding zitten als facultatieve elementen aan het sjabloon toegevoegd worden.

Om een sjabloon te creëren worden in de eerste stap twee voorbeelden veralgemeend tot een sjabloon. In de volgende stappen wordt het sjabloon telkens veralgemeend door het wijzigingsscript van de afbeelding met een nieuw voorbeeld te zoeken en de voorgaande regels toe te passen. Om een sjabloon te creëren wordt dus eerst het volledige wijzigingsscript voor twee bomen gezocht. Nadat het optimale wijzigingsscript bekend is kan de eerste boom als basis genomen worden en kunnen de wijzigingen toegepast worden, door de deelbomen te snoeien op de plaats waar een wijziging aangegeven staat in het script; deze knoop wordt als joker aangeduid. Voor invoeringen moet een extra jokerknoop aan de boom toegevoegd worden met het label van de knoop uit de tweede boom. Dit gebeurt in een stap achteraf en betekent dat er dus over beide bomen geïtereerd moet worden. Deze manier wordt gebruikt door Hogue [6] maar hij vermeldt de invoeging van jokerknopen niet. Hierdoor bestaat het gevaar dat deelbomen die enkel in het nieuwe voorbeeld voorkomen niet gerepresenteerd zullen worden.

4.2 Prijsfuncties

Voor het bepalen van de boomwijzigingsafstand wordt telkens de goedkoopste afbeelding gekozen. De prijs wordt berekend op basis van het aantal knopen dat ingevoegd, verwijderd dan wel vervangen moet worden. Volgens definitie 7 heeft elke wijziging een eenheidsprijs.

4.2.1 Inertie

Een joker kan meer dan één element van een boom representeren: ook afstammelingen van de knoop waarvan de joker in de plaats komt worden door hem gerepresenteerd. Door het gebruik van een eenheidsprijs voor elke wijziging heeft de wijziging van een joker altijd een prijs van één. Zo bestaat natuurlijk het gevaar dat een joker nooit in de afbeelding zit omdat het goedkoper is hem te verwijderen en een andere knoop in te voegen dan om hem uit te lijnen met een grote boom. Reis

en Hogue vermelden dit probleem niet. Hogue heeft hier waarschijnlijk geen last van omdat hij maar met heel kleine bomen werkt en twee of drie leervoorbeelden meestal volstaan om een sjabloon te creëren. Omdat voor het leren van grote bomen meer leervoorbeelden nodig zijn kan er hier niet aan voorbij gegaan worden.

Om dit probleem tegen te gaan heb ik het concept *inertie* geïntroduceerd; dit geeft aan dat het duurder is om nieuwe jokers te creëren. De uitlijning van een knoop krijgt de voorkeur boven de wijziging van een knoop. Dit betekent dat de eenheidsprijs voor een wijziging niet meer volstaat wanneer sjablonen iteratief geconstrueerd worden. Met andere woorden, wijzigingen moeten duurder worden dan uitlijning met een joker, wat weer duurder moet zijn dan gelijke knopen die geen jokers zijn. Dit geeft aanleiding tot de volgende definitie:

Definitie 11 (Prijs van een individuele wijziging met inertie)

$$\gamma(v \rightarrow w) = \begin{cases} 0 & \text{als } v = w \\ 1 & \text{voor elke knoop in } w, \text{ als } w \text{ uitgelijnd wordt door joker } v \\ 10 & \text{als } v \neq w \end{cases}$$

De gelijkheidsdefinitie blijft hier onveranderd van toepassing: als de labels hetzelfde zijn zijn ook de knopen hetzelfde. Met deze prijsfunctie wordt telkens getracht zo veel mogelijk van de bomen (structureel) gelijk te houden en pas jokers in te voegen als het echt niet anders kan.

4.2.2 Vervangingen mogen niet altijd

Een ander probleem wat niet door Reis en Hogue vermeld wordt is de speciale status van bladeren. De voor informatievergarig interessante verschillen zitten immers in of nabij de bladeren van de bomen. Dit betekent dat we voor alle knopen die geen bladeren zijn liever een verwijdering en een invoeging zien dan de vervanging van dat element. Dit is ook logisch aangezien een element hoger in de boom veel waarschijnlijker een element voor de opmaak van het document en dus inhoud is. Single- en plusjokers moeten immers de interessante delen van een pagina representeren en kunnen alleen maar door (combinatie met) een vervanging gecreëerd worden. Ook op een andere manier speelt de definitie van jokers hier een rol. Jokers kunnen maar één soort element representeren aangezien een joker een gewone knoop met een speciaal attribuut is. Als er een vervanging, in plaats van een invoeging en een verwijdering, zou plaatsvinden op een knoop die geen blad is, zou een joker

twee verschillende labels moeten hebben. Dit is niet mogelijk volgens de HTML-specificaties. Aangezien binnen het HTML-domein de interessante verschillen alleen maar in de bladeren zitten en de jokers maar met één element kunnen uitlijnen, zullen we de vervangingen een aparte status in de prijsfunctie moeten geven. We staan vervangingen alleen toe op bladeren en maken vervangingen daarom duurder. De nieuwe prijsfunctie, gecombineerd met het concept van inertie, valt dan uiteen in twee delen:

Definitie 12 (Verwijdering en invoeging met inertie)

$$\gamma(v \rightarrow w) = \begin{cases} 1 & \text{voor elke knoop in } w, \text{ als } w \text{ uitgelijnd wordt door joker } v \\ 10 & \text{als } v \neq w \end{cases}$$

Definitie 13 (Vervanging met inertie)

$$\gamma(v \rightarrow w) = \begin{cases} 0 & \text{als } v = w \\ 1 & \text{voor elke afstammeling van } w, \text{ als } w \text{ uitgelijnd wordt door joker } v \\ 100 & \text{als } v \neq w \end{cases}$$

De prijzen in deze functies blijven dezelfde als voor een individuele wijziging.

4.3 Nabewerkingen

Bij het leren van een sjabloon werken we maar met een klein aantal voorbeelden. Het is de bedoeling dat die met zo veel mogelijk bomen uit zijn domein (één blog- of nieuwssite) uitlijnt. Vaak is het echter zo dat er variabele voorkomens van elementen zijn zoals bij lijsten. Als er maar met één soort jokers gewerkt wordt zal het sjabloon de voorbeelden wel goed representeren, maar bestaat het gevaar dat lijsten met een groter aantal voorkomens niet uitgelijnd kunnen worden. Het is de bedoeling dat alle elementen uit deze lijst uitgelijnd worden en niet alleen het aantal dat toevallig in de voorbeelden voorkwam.

Hogue ontwikkelde hiertoe een mechanisme dat hij ‘list collapse’ noemde. In een nabewerkingsstap (post-processing) wordt in het sjabloon voor alle buurknopen met hetzelfde label de boomwijzigingsafstand berekend. Als de prijs daarvan lager ligt dan een bepaalde drempelwaarde — in het geval dat de deelbomen erg op elkaar lijken en de afbeelding dus een lage prijs heeft, zullen de bomen in elkaar geschoven worden. Voor twee deelbomen met als oorsprong knopen v en w en een afbeelding M_{vw} worden dan eerst de knopen van deelboom v die verwijderd worden

in M_{vw} , vervangen door jokers. Daarna wordt de hele deelboom w , inclusief de oorsprong, verwijderd. Tijdens de extractie worden alle buurknopen met hetzelfde label uitgelijnd. Het mag duidelijk zijn dat deze nabewerkingsstap erg duur is en dus niet praktisch is voor grote bomen.

Reis loste het probleem van de variabele lijsten op door verschillende soorten jokers te gebruiken. In een nabewerkingsstap kunnen jokers die naast elkaar liggen en hetzelfde label hebben gecombineerd worden tot een plus of kleene, zoals gedefinieerd in hoofdstuk 4.1.2.

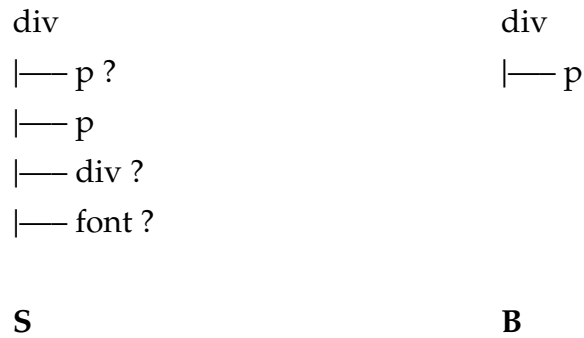
Tijdens de analyse van de experimenten is me opgevallen dat er vaak normale knopen zijn waarvan alle kinderknopen jokerknopen zijn. Omdat compacte representaties sneller en duidelijker zijn heb ik ook geëxperimenteerd met het *snoeien* (prunen) van dat soort deelbomen. We verschuiven de jokers dan van de kinderknopen naar de ouder met een gecombineerde jokeraanduiding. Zoals blijkt uit de resultaten is deze stap zeer zinvol.

Al deze nabewerkingsstappen zorgen er voor dat we een kleiner, compacter sjabloon krijgen dat met meer pagina's van dezelfde site kan uitlijnen.

4.4 Extractie

Als er eenmaal een sjabloon gecreëerd is kan dit gebruikt worden om de interessante informatie uit gelijkaardige webpagina's te extraheren. Extractie gebeurt door een sjabloon uit te lijnen en de deelbomen van die knopen die uitgelijnd worden met een joker uit de boom te lichten. In principe kan de boomwijzigingsafstand gebruikt worden voor het extraheren van de sjablonen, als de boomwijzigingsafstand het sjabloon en de pagina uitlijnt met een lage prijs. Gebruik van de boomwijzigingsafstand is echter erg prijzig en dus heb ik een snellere en simpelere manier bedacht.

Aangezien we weten dat een extractie pas slaagt als er sprake is van een uitlijning, als er een afbeelding zonder wijzigingen is, hoeven we niet de boomwijzigingsafstand tussen alle knopen te berekenen om te weten op welke knopen we dieper de boom in moeten gaan. Voor elke twee deelniveaus waarvan de ouders in de afbeelding zitten, waarvan we dus zeker weten dat de ouders gelijk zijn, vergelijken we een representatie van dat deelniveau. Deze representatie wordt verkregen door de labels van alle knopen van dat deelniveau, zonder eventuele kinderknopen, achter elkaar te zetten in een string. Er wordt dus een afvlakking (flattening) van de knopen toegepast maar alle informatie over de volgorde en de labels van de knopen, beno-



Figuur 4.2: Sjabloon S en Boom B.

digd voor de uitlijning van twee deelniveaus, blijft behouden. De strings kunnen dan vergeleken worden met behulp van een reguliere expressie. De jokerknopen zijn de jokers van de reguliere expressie zijn worden als volgt gerepresenteerd: een *single* moet 1 knoop extraheren en wordt dan ook aangeduid als *(label)*, een *optional* mag 0 of 1 knoop extraheren en wordt aangeduid als *(label)?*, een *plus* moet minstens 1 knoop extraheren en wordt aangeduid als *(label)+* en tot slot wordt een *kleene* die 0 of meerdere knopen mag extraheren *(label)**. Voor de knopen uit het voorbeeldje van Figuur 4.2 wordt het deelniveau van *S* als $(p)?(p)(div)?(font)?$ gerepresenteerd en dat van *B* als p . Via de ingebouwde *preg_match*-functie van PHP wordt dan de volgende array weergegeven als het resultaat van de toepassing van de reguliere expressie: $\text{array}(0=>,1=>p,2=>,3=>)$. Dit kunnen we zo interpreteren dat alleen het tweede element uit sjabloon *S*, dus de tweede *p*, uitgelijnd wordt en de rest in het resulterende sjabloon facultatief en niet vindbaar is. Mocht een facultatief element (optional of kleene) wel uitlijnen dan zou dit in de array weergegeven worden. Ook hier wordt alleen gerecurseerd op knopen die geen jokers zijn. Als er geen uitlijning gevonden wordt tussen de reguliere expressie en de string zal het algoritme eindigen met een foutmelding.

Hoe snel dit gulzige (greedy) algoritme een uitlijning vindt hangt af van de grootte van het sjabloon, wat resulteert in een tijdsduur van $O(|S|)$. Ook het vinden van de uitlijning van de reguliere expressie moet in rekening gebracht worden. Dit is $O(nm)$, waar n de grootte van de string is en m de grootte van de reguliere expressie.¹ In ons

¹Dit steunt op een resultaat uit de formele taaltheorie die stelt dat elke niet-deterministische eindige toestandsautomaat (NFA) omgezet kan worden in een deterministische eindige toestandsautomaat (DFA). Het algoritme simuleert deze omzetting en past de zo verkregen DFA toe op de inputstring, symbool per symbool. Dit laatste proces heeft een tijd lineair met de grootte van de inputstring.

geval is de slechtste tijd hiervoor dus twee maal de maximale vertakkingsgraad van de boom. Ook moet opgemerkt worden dat jokers met meerdere knopen uitgelijnd kunnen worden. We moeten bij het bepalen van de grootte van het sjabloon dus het maximaal aantal buurknopen waarmee een bepaalde joker kan uitlijnen in acht nemen.

Voor de vertaalslag van jokers naar een reguliere expressie houden we bij de nabewerking in de jokerknoop bij hoeveel knopen er maximaal met deze knoop uitgelijnd kunnen worden, gegeven onze voorbeelden. Vooralsnog gebruik ik deze informatie ook om te bepalen hoeveel elementen een bepaalde joker mag uitlijnen in de reguliere expressie. Dit is natuurlijk niet heel flexibel voor bijvoorbeeld lijsten met een variabele lengte en wordt later ook aangepast. Het maakt de experimenten en de analyses echter wel gemakkelijker .

4.5 Samenvatting

In dit hoofdstuk is besproken hoe sjablonen geconstrueerd kunnen worden met behulp van de gevonden afbeelding, door de berekening van de beperkte top-down boomwijzigingsafstand. Er is uitgelegd dat door de vervanging van hele deelbomen door jokers een deel van de informatie benodigd voor de berekening van de boomwijzigingsafstand verloren gaat. Dit is simpel op te lossen door de prijsfunctie aan te passen, zodat het moeilijker wordt om nieuwe jokers te creëren. Het feit dat de veranderingen voornamelijk in de bladeren voorkomen en dat jokers maar een soort element kunnen representeren heeft tot een verdere aanpassing van de prijsfunctie geleid zodat vervangingen alleen maar in de bladeren zullen gebeuren. Dit scherpt de top-down definitie dus aan. In de laatste sectie is een snellere manier beschreven dan het gebruik van de boomwijzigingsafstand om de door de jokerknopen gerepresenteerde informatie te extraheren. Deze manier van uitlijning dient ook als basis voor de benadering van de beperkte top-down boomwijzigingsafstand die in het volgende hoofdstuk beschreven wordt.

Precieser kan gesteld worden dat een inputstring met grootte n getest kan worden in vergelijking met een reguliere expressie van grootte m in een tijd van $O(n+2m)$ of $O(nm)$, naargelang de implementatie.

Hoofdstuk 5

Een benadering van de beperkte top-down boomwijzigingsafstand

In de vorige hoofdstukken zijn verschillende varianten van de boomwijzigingsafstand besproken. In dit hoofdstuk wordt vastgesteld dat ook de beperkte top-down boomwijzigingsafstand nog steeds zeer prijzig is. Om de complexiteit te verminderen heb ik een algoritme geïmplementeerd dat een benadering is van de optimale afbeelding van de beperkte top-down boomwijzigingsafstand.

5.1 De hoge prijs voor de berekening van de boomwijzigingsafstand

Binnen ons domein, een verzameling automatisch gegenereerde HTML-pagina's van dezelfde site, bevinden de verschillen van de bomen zich in of nabij de bladeren van de bomen. In deze top-down aanpak is de kans groter dat de bomen verschillen naarmate je dichter bij de bladeren komt. Hoe hoger je echter in de boom zit, hoe groter de kans dat de elementen van de twee bomen gelijk zijn. Dit betekent ook dat als twee knopen hoog in de boom niet hetzelfde label hebben er eerder een invoeging of verwijdering moet gebeuren dan een vervanging.

Bij de berekening van de beperkte top-down afbeelding wordt hiermee rekening gehouden, door hele deelbomen te verwijderen of in te voegen als een blad vergeleken wordt met een boom. Als twee knopen hetzelfde label hebben zullen er recursief afbeeldingen tussen al hun kinderknopen en afstammelingen berekend worden — om te kijken welke knopen en hun deelbomen zo goedkoop mogelijk met elkaar

uitgelijnd kunnen worden. De beperkte top-down boomwijzigingsafstand berekent dus de afbeelding voor elke knoop met hetzelfde label; waarvan bekend is dat de ouders in de afbeelding zitten. Intuïtief is het duidelijk dat als meerdere knopen van een bepaald deelniveau hetzelfde label hebben, maar een van de berekende afbeeldingen tussen die knopen de optimale zal zijn.

Het zou handig zijn om een andere maat te hebben die kan vertellen of twee bomen, waarvan de oorsprong hetzelfde label heeft, gelijkaardig zijn. In plaats van te recursen over deelbomen heb ik dan ook geprobeerd de grootte van die deelbomen te nemen als maat voor gelijk(aardig)heid. Deze maat werkt het beste boven in een boom, aangezien bij vergelijking de grootte van een boom dan ook veel zegt over de structuur van die boom. Ook van deelbomen nabij bladeren wordt verwacht dat dit een goede maat is. Deelbomen die niet op dezelfde plaats staan hebben immers wel vaak een oorsprong met hetzelfde label, maar een verschillende semantische betekenis, en dus vaak grootte. In plaats van de structuur van de boom te vergelijken wordt dus een benadering van de kleinste prijs van de beperkte top-down afbeelding gemaakt door de grootte van de bomen te vergelijken. Uiteindelijk blijkt dat de grootte van de deelbomen niet de beste maat is. De beste resultaten worden verkregen door de voorkeur te geven aan groepen van gelijke elementen. In de volgende secties leg ik beide uit en bekijk ik waar het goed gaat en waar niet.

5.2 Een algoritme dat de beperkte top-down afbeelding benadert

Mijn benadering van de berekening van de optimale beperkte top-down afbeelding gebeurt door de wijzigingsafstand voor strings toe te passen op de labelrepresentatie van een deelniveau van twee bomen. De prijs voor een wijziging hangt echter af van de grootte van de boom, gerepresenteerd door zijn oorsprong met een bepaald label.

In plaats van, zoals bij de beperkte boomwijzigingsafstand, te recursen over alle combinaties van knopen met hetzelfde label op hetzelfde deelniveau in twee bomen, wordt alleen gerecurseerd over knopen met hetzelfde label waarvan redelijkerwijs vermoed wordt dat ook hun afstammelingen in de afbeelding zitten, zonder dit expliciet te berekenen. In Figuur 5.1 staat de pseudocode voor het algoritme dat dit implementeert.

```

1 Benadering( B1, B2 )
2   l1 = |kinderen( B1 )|
3   l2 = |kinderen( B2 )|
4   if( B1 != B2 )
5     voeg B1 toe aan het sjabloon en markeer hem optioneel
6     voeg B2 toe aan het sjabloon en markeer hem optioneel
7   elseif( l1 = 0 & l2 != 0)
8     voeg kinderen( B2 ) toe aan het sjabloon en markeer ze optioneel
9   elseif( l1 != 0 & l2 = 0 )
10    voeg kinderen( B1 ) toe aan het sjabloon en markeer ze optioneel
11  else
12    if( VERTAAL( B1 ) == VERTAAL( B2 ) )
13      for i = 0 to l1 /* in dit geval is l1 = l2 */
14        WAB( B1[i], B2[i] )
15      end
16    else
17      wijzigingen = Wijzigingsafstand( VERTAAL( B1 ), VERTAAL( B2 ) )
18      i = j = 0
19      while( ( i < l1 ) || ( j < l2 ) )
20        if( wijzigingen[i][j] == verwijdering )
21          voeg knoop B1[i] toe aan het sjabloon en markeer hem optioneel
22          j-- /* i is verwijderd */
23        elseif( wijzigingen[i][j] == invoeging )
24          voeg knoop B2[j] toe aan het sjabloon en markeer hem optioneel
25          i-- /* j is ingevoegd */
26        elseif( wijzigingen[i][j] == verandering )
27          voeg knoop B1[i] toe aan het sjabloon en markeer hem single
28        elseif( wijzigingen[i][j] == zelfde Label )
29          voeg knoop B1[i] toe aan het sjabloon
30          Benadering( B1[i], B2[j] )
31        /* else er zijn geen wijzigingen gedefinieerd */
32        end
33        i++, j++
34      end
35    end
36  end
37 end
38
39 VERTAAL( B )
40   string = zet de labels van kinderen( B ) achter elkaar in een string
41   return string
42 end

```

Figuur 5.1: De pseudocode voor de wijzigingsafstand in bomen.

In regel 4 tot en met 6 van Figuur 5.1 wordt eerst gekeken of de oorsprong van de bomen hetzelfde is. Zoniet kunnen we gelijk de ene boom verwijderen en de andere invoegen. In regel 7 tot en met 10 wordt bepaald of een van de knopen een blad is. Zo ja moet de andere knoop als een facultatieve jokerknoop ingevoegd worden in het sjabloon. Regel 4 tot en met 10 zijn dus dezelfde shortcuts als die van de beperkte top-down afbeelding.

Aangezien we willen — en weten — dat zo veel mogelijk van de bomen structureel gelijk blijft berekenen we niet de afstand tussen alle combinaties van knopen met hetzelfde label van twee deelniveaus, maar alleen van twee deelniveaus waarvan de knopen verschillen en de ouders in de afbeelding zitten, waarvan we dus zeker weten dat de ouders gelijk zijn. Als eerste verbetering berekenen we voor twee deelniveaus niet de hele (boom)wijzigingsafstand maar vergelijken we de stringrepresentatie ervan. Dit wordt verkregen door de labels van alle knopen van dat deelniveau, zonder eventuele kinderknopen, achter elkaar te zetten in een string. Deze strings behouden alle informatie over de volgorde en de labels van de knopen en kunnen simpel vergeleken worden. We passen hier dus weer de afvlakking zoals besproken in het vorige hoofdstuk toe. Als de strings hetzelfde zijn kunnen we voor elke in die string gerepresenteerde knoop afdalen in de boom. Dit wordt toegepast in regel 12 tot en met 15, waar vastgesteld wordt of de stringrepresentatie van de bepaalde deelniveaus gelijk is. Zo ja wordt voor elk knopenpaar $(v[i], w[i])$ gerecurseerd waarbij $v[i]$ en $w[i]$ de knopen van respectievelijk deelniveau (v) en deelniveau (w) zijn. Merk op dat doordat de stringrepresentaties van de deelniveaus gelijk zijn, $\text{graad}(v)$ gelijk is aan $\text{graad}(w)$. Juist omdat in ons domein de bomen op een top-down manier hetzelfde zijn versnelt deze stap het hele proces aanzienlijk.

Als de twee strings niet gelijk zijn, bestaan er verschillen tussen de deelniveaus. In dit geval passen we op regel 17 een benadering van de beperkte top-down boomwijzigingsafstand toe, om de afbeelding tussen die twee deelniveaus te vinden. We berekenen dan de gewone wijzigingsafstand op de stringrepresentatie van die bepaalde deelniveaus, maar met een aparte prijsfunctie die in de volgende sectie wordt uitgelegd. We vermijden zo dat we voor het bepalen van de afbeelding moeten recursen op elke knoop met hetzelfde label; in plaats van de minimale afbeelding tussen alle deelbomen met hetzelfde label wordt er een andere maat van gelijkheid gebruikt. Stel $m = \text{graad}(v)$ en $n = \text{graad}(w)$. We creëren dan een $m + 1$ bij $n + 1$ matrix D . $D[i][j]$ bevat de afbeelding M_{ij} tussen knopen $v[i]$ en $w[j]$, met $0 \leq i \leq m + 1$

en $0 \leq j \leq n + 1$. We kunnen dan de cellen van D berekenen door vast te stellen dat:

$$D[i][j] = \min_{\gamma} \begin{cases} D[i][j-1] & + \text{ invoegprijs}(w[j]) \\ D[i-1][j] & + \text{ verwijderprijs}(v[i]) \\ D[i-1][j-1] & + \text{ vervangingsprijs}(v[i], w[j]). \end{cases}$$

$D[0][0]$ is gedefinieerd als een lege afbeelding. De prijs van de wijzigingen — en gelijkheid — wordt in de volgende sectie besproken.

Door de achterwaartse gang in de zo geconstrueerde prijsmatrix kan het wijzigingsscript voor de deelniveaus gevonden worden. Dat wijzigingsscript wordt toegepast op regel 20 tot en met 27 in Figuur 5.1. In het wijzigingsscript wordt nu ook bijgehouden welke knopen volgens de nieuwe prijsfunctie hetzelfde blijven. Alleen voor deze knopen wordt verder gerecurseerd en het hele proces opnieuw toegepast. Merk ook op dat in regel 22 en 25 rekening gehouden wordt met knopen die zijn ingevoegd of verwijderd voor de plaatsbepaling in het bepaalde deelniveau in de bomen.

5.3 De prijsfunctie voor de benadering

Net als bij de beperkte top-down boomwijzigingsafstand moeten we rekening houden met reeds ingevoegde jokers en met het feit dat vervangingen alleen in de bladeren mogen gebeuren. Als aanvulling hierop kan gesteld worden dat tekstknopen extra belangrijk zijn. Het label van een tekstknoop is altijd '#text' in de DOM-representatie. De inhoud van die knoop, de eigenlijke tekst, wordt dan door de gelijkheidsdefinitie van knopen over het hoofd gezien. Deze houdt echter vaak een goede structurele markering voor een wrapper in. Het herleiden van dit probleem tot de gewone wijzigingsafstand voor strings laat dit duidelijk zien. Stel string 1 = 'xyz' en string 2 = 'aby'. Als de prijs voor het vervangen van elk karakter één is dan zal 'xyz' omgezet worden in 'aby' met een prijs van drie, door gewoon elk karakter van string 1 te vervangen. Als we in plaats van karakters echter knopen uit een HTML-boom beschouwen willen we dat alles structureel zo veel mogelijk hetzelfde blijft en dan hebben we liever 'x?a?b?yz?' als oplossing. De knoop 'y' is immers hetzelfde in alle voorbeelden. Als alle knopen uit de strings tekstknopen zijn zullen de labels van deze strings allemaal hetzelfde zijn: '#text#text#text'. Dit betekent dat we vervangingen niet alleen op bladeren moeten toestaan maar ook de inhoud van deze bladknopen tijdens de vergelijking in acht moeten nemen. Een

extra voordeel van deze benadering is dat het veel onwaarschijnlijker is dat tekst hetzelfde is dan elementen uit de beperkte HTML-markeringen. Er zijn immers veel meer mogelijke stringcombinaties dan elementen in HTML. Als de tekst hetzelfde is in twee webpagina's is het goed mogelijk dat dit een 'structureel' belangrijk element is. Daarom moeten we het uitlijnen van tekstknopen goedkoper maken dan het uitlijnen van andere knopen.

Omdat er niet gerecurseerd wordt als twee knopen hetzelfde zijn hebben we een andere maat nodig die aangeeft of hun deelbomen hetzelfde zijn. Ik heb hiervoor verschillende soorten prijsfuncties uitgetest. Als eerste heb ik het absolute verschil in grootte van de bomen geworteld in die knopen genomen. Dit betekent dat bomen die op elkaar lijken, waarbij het verschil in grootte op het verschil in structuur wijst, een lagere prijs hebben dan bomen die niet op elkaar lijken. Er wordt verwacht dat dit een goede maat is, aangezien deelbomen die niet op dezelfde plaats staan wel vaak een oorsprong met hetzelfde label hebben maar in semantische betekenis, en dus meestal grootte, verschillen.

Dit alles samen leidt tot een hiërarchische prijsstructuur die weer uit twee delen bestaat:

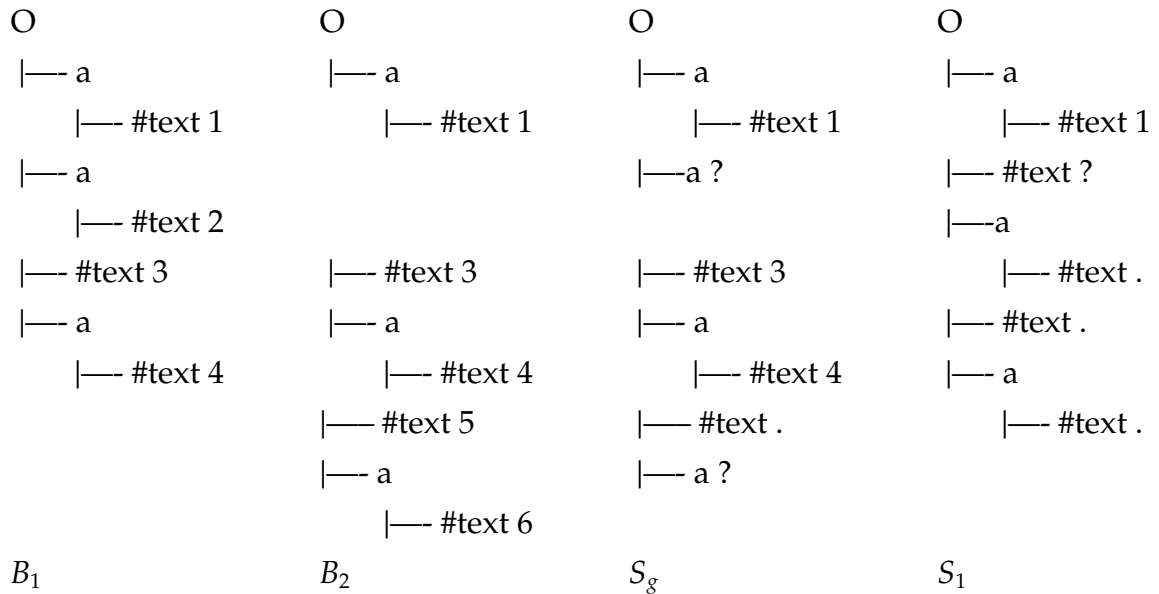
Definitie 14 (Verwijdering en invoeging voor de benadering)

$$\gamma(v \rightarrow w) = \begin{cases} 10 & \text{voor elke knoop in } w, \text{ als } w \text{ uitgelijnd wordt door joker } v \\ 100 & \text{als } v \neq w \end{cases}$$

Definitie 15 (Vervanging met inertie voor de benadering)

$$\gamma(v \rightarrow w) = \begin{cases} 0 & \text{als } inhoud(v) = inhoud(w) \\ & \text{en } v \text{ en } w \text{ tekstknopen zijn} \\ 10 & \text{als } inhoud(v) \neq inhoud(w) \\ & \text{en } v \text{ en } w \text{ tekstknopen zijn} \\ 10 * |F(v) - F(w)| & \text{als } v = w \\ 10 & \text{voor elke afstammeling van } w, \\ & \text{als } w \text{ uitgelijnd wordt door joker } v \\ 1000 & \text{als } v \neq w \end{cases}$$

Helaas blijkt uit experimenten dat als de naburige knopen geen tekstknopen zijn, de aanpak voor tekst als structurele markering niet werkt. Het blijft immers goedkoper om een bladknoop te verwijderen of in te voegen dan om een knoop met afstammelingen te verwijderen of in te voegen.

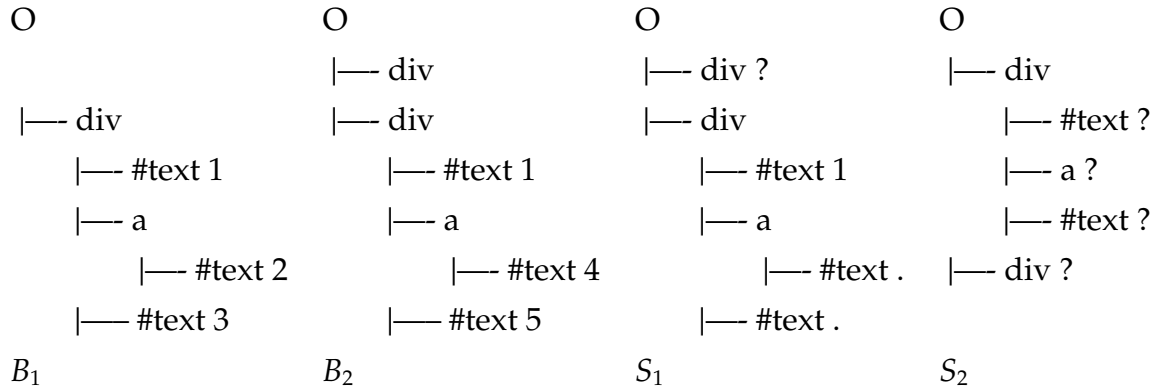


Figuur 5.2: Gewenst sjabloon S_g en sjabloon met de nieuwe prijsfunctie S_1 .

Alleen het verschil in grootte van de bomen blijkt geen goede aanduiding te zijn. Bekijk de bomen in Figuur 5.2. We willen van B_1 en B_2 het gewenste sjabloon S_g verkrijgen; toch verkrijgen we sjabloon S_1 vanwege een eigenschap van de wijzigingsafstand voor strings. Let wel: we vergelijken alleen de kinderen van oorsprong O . We bepalen dus de wijzigingsafstand, met bovenstaande prijsfunctie, tussen de strings ‘aa#texta’ en ‘a#texta#texta’, waarbij we voor de ‘#text’ knopen ook naar de inhoud kijken. Omdat voor de bepaling van het wijzigingsscript vertrokken wordt van de laatste elementen van de strings lijkt het alsof de laatste ‘a’ in B_1 hetzelfde is als de laatste ‘a’ uit B_2 . Het goedkoopste daarna is het vervangen van ‘#text 3’ door ‘#text 5’. Dan zal de voorlaatste ‘a’ uit B_1 uitlijnen met de voorlaatste ‘a’ uit B_2 . Ten slotte lijkt de ‘#text 3’ uit B_2 facultatief en lijnen de eerste ‘a’s met elkaar uit.

Merk op dat ditzelfde probleem ook voorkomt bij de beperkte top-down afbeelding. Daar wordt weliswaar gerecurseerd tot in de tekstknopen die kinderen zijn van de ‘a’s, maar daarna is het goedkoper om de kinderen te vervangen dan om de laatste ‘a’ uit de tweede boom in te voegen. Omdat ik dit probleem juist wil heb ik een andere prijsfunctie bedacht, die ook bij de experimenten gebruikt zal worden.

Een prijsfunctie waarbij getracht wordt om naburige elementen met eenzelfde label bij elkaar te houden blijkt hiervoor de oplossing. In deelniveaus die van elkaar verschillen heeft een element met een ander label dan zijn omringende elementen immers een grotere kans om structureel onderscheidend te zijn. Hierbij laten we de



Figuur 5.3: Sjablonen gecreërd met prijsfunctie 1 & 2.

grootte van de bomen helemaal buiten beschouwing. Dit kan als volgt gedefinieerd worden:

Definitie 16 (Verwijdering en invoeging voor de benadering)

$$\gamma(v \rightarrow w) = \begin{cases} 0 & \text{als } v = w \\ 10 & \text{voor elke knoop in } w, \text{ als } w \text{ uitgelijnd wordt door joker } v \\ 100 & \text{als } v \neq w \end{cases}$$

Definitie 17 (Vervanging met inertie voor de benadering)

$$\gamma(v \rightarrow w) = \begin{cases} 0 & \text{als } v = w \\ 0 & \text{als } inhoud(v) = inhoud(w) \text{ en } v \text{ en } w \text{ tekstknopen zijn} \\ 10 & \text{als } inhoud(v) \neq inhoud(w) \text{ en } v \text{ en } w \text{ tekstknopen zijn} \\ 10 & \text{voor elke afstammeling van } w, \\ & \text{als } w \text{ uitgelijnd wordt door joker } v \\ 1000 & \text{als } v \neq w \end{cases}$$

Bij deze prijsfunctie wordt zo veel mogelijk getracht om elementen met hetzelfde label bij elkaar te houden — vandaar ook de gelijkheidstest bij verwijderingen en invoegingen. Het heeft hier ook geen zin meer om de grootte van de bomen bij te houden omdat de prijs van de verwijderingen of invoegingen van elementen met hetzelfde label toch 0 is. Merk op dat deze prijsfunctie voor de originele beperkte top-down afbeelding niet zou werken, omdat daar de invoegingen en verwijderingen alleen maar bekeken worden als er vergeleken wordt met een blad.

Een voorbeeld waar deze laatste prijsfunctie fout gaat staat in Figuur 5.3. Het sjabloon S_1 is gecreërd met de eerste prijsfunctie beschreven in dit hoofdstuk, S_2 met

de tweede prijsfunctie. Met deze laatste kost het niets om een element met hetzelfde label in te voegen of te verwijderen. Daarom wordt de eerste 'div' uit B_1 uitgelijnd met de 'div' uit B_2 . De laatste 'div' wordt verwijderd. Doordat het algoritme denkt dat de eerste 'div' uit B_2 het gelijke element is worden alle kinderen van de 'div' uit B_1 verwijderd en dus als facultatief gemerkt. Dit sjabloon extraheert wel nog de juiste deelbomen, maar er gaan structurele eigenschappen verloren.

Doordat in de eerste prijsfunctie de grootte van de bomen meegenomen wordt, wordt hiermee wel de goede uitlijning gevonden voor het voorbeeld in Figuur 5.3. Omgekeerd levert de tweede prijsfunctie wel het gewenste sjabloon S_g in Figuur 5.2 op. Beide prijsfuncties hebben dus hun eigen voor- en nadelen maar omdat de eerste prijsfunctie gelijkaardige problemen heeft als de beperkte top-down boomwijzigingsafstand zal alleen de tweede prijsfunctie gebruikt worden voor de experimenten in het volgende hoofdstuk.

Een mogelijke oplossing voor het probleem van de tweede prijsfunctie kan gevonden worden in grammaticale inductie. Daar blijkt dat als de knopen opnieuw gelabeld worden zodat ze ook het label van hun kinderen meedragen, een verbetering tot 50% mogelijk is. Door tijdgebrek heb ik dit helaas niet kunnen testen.

5.4 Complexiteit

In een eerste stap berekenen we voor elke knoop in de bomen de grootte van de deelboom met deze knoop als oorsprong. Dit heeft een complexiteit die lineair is met de grootte van de bomen.

Ook de ondergrens aan de complexiteit van mijn algoritme is lineair met de grootte van de boom. Als twee bomen volledig hetzelfde zijn is de snelheid van mijn implementatie gelijk aan de grootte van boom $|B_1| = |B_2|$ en de ondergrens aan de complexiteit dus $\theta(|B_1|)$.

De bovengrens van mijn algoritme is $O(|B_1| \cdot |B_2|)$, maar dit is enkel het geval als de bomen een diepte van 2 hebben en alle knopen dus blaadjes zijn. Dit komt bij HTML-bomen echter nooit voor. Het feit dat we alleen recursen als de labels gelijk zijn, geeft aan dat de bovengrens eerder in de buurt ligt van $O(|L_1| \cdot |L_2|)$, aangezien alleen die deelniveaus vergeleken worden waarvan de bladeren verschillen. Om de verbetering te toetsen is een aantal proeven gedaan dat de verschillende afbeeldingen met elkaar vergelijkt. De resultaten hiervan staan in het volgende hoofdstuk.

5.5 Samenvatting

In dit hoofdstuk heb ik aangetoond dat de top-downaspecten van ons domein gebruikt kunnen worden voor een snellere berekening van de beperkte top-down boomwijzigingsafstand. Dit gebeurt door stringrepresentaties van de deelniveaus te vergelijken. Ook heb ik geprobeerd om de vele recursies die de beperkte top-down boomwijzigingsafstand nodig heeft voor knopen met gelijke labels op dezelfde deelniveaus te verminderen. Dit is geïmplementeerd door alleen de wijzigingsafstand te berekenen van de stringrepresentaties met een nieuwe prijsfunctie. De prijs die in mijn implementatie berekend wordt is een benadering van de beperkte top-down afstand en zegt behalve voor het vinden van een afbeelding niets over de relatie tussen de twee bomen.

Hoofdstuk 6

Experimentele resultaten

Om de beweringen en algoritmes in deze scriptie te testen heb ik verschillende experimenten uitgevoerd. In deze scriptie heb ik betoogd dat de verschillen tussen een verzameling gerelateerde HTML-pagina's in of nabij de bladeren van de boomrepresentaties van die pagina's zit. Voor het vinden van die verschillen kan gebruik gemaakt worden van de beperkte boomwijzigingsafstand. Ik heb gesteld dat deze benaderd kan worden door slim gebruik van de eigenschappen van ons domein. In hoofdstuk 5.4 heb ik beredeneerd dat mijn benadering sneller zal zijn dan de volledige berekening van de beperkte boomwijzigingsafstand maar het bleek moeilijk om precies te bepalen hoe veel sneller. Daarom zal empirisch de snelheid van de verschillende boomwijzigingsafstanden tegen elkaar afgezet worden.

Ook de via de verschillende methodes gecreëerde sjablonen zijn met elkaar vergeleken. Voor deze vergelijkingen heb ik de beperkte top-down boomwijzigingsafstand tegen mijn benadering afgezet. Er wordt natuurlijk getest of met behulp van de verschillende besproken boomwijzigingsafstanden een nuttige veralgemening van een verzameling voorbeelden gecreëerd kan worden. De kwaliteit hiervan wordt bij machine learning gedefinieerd in termen van precisie. Uit de tests blijkt dat mijn methode meestal een goede benadering van de sjablonen zoals gecreëerd door Reis [5] oplevert, maar niet altijd. Hier zal uitgebreid op ingegaan worden.

Andere interessante vragen zijn het aantal benodigde leervoorbeelden voor het verkrijgen van een goed sjabloon en of de resultaten op de testpagina's verbeterd kunnen worden door nabewerkingsstappen.

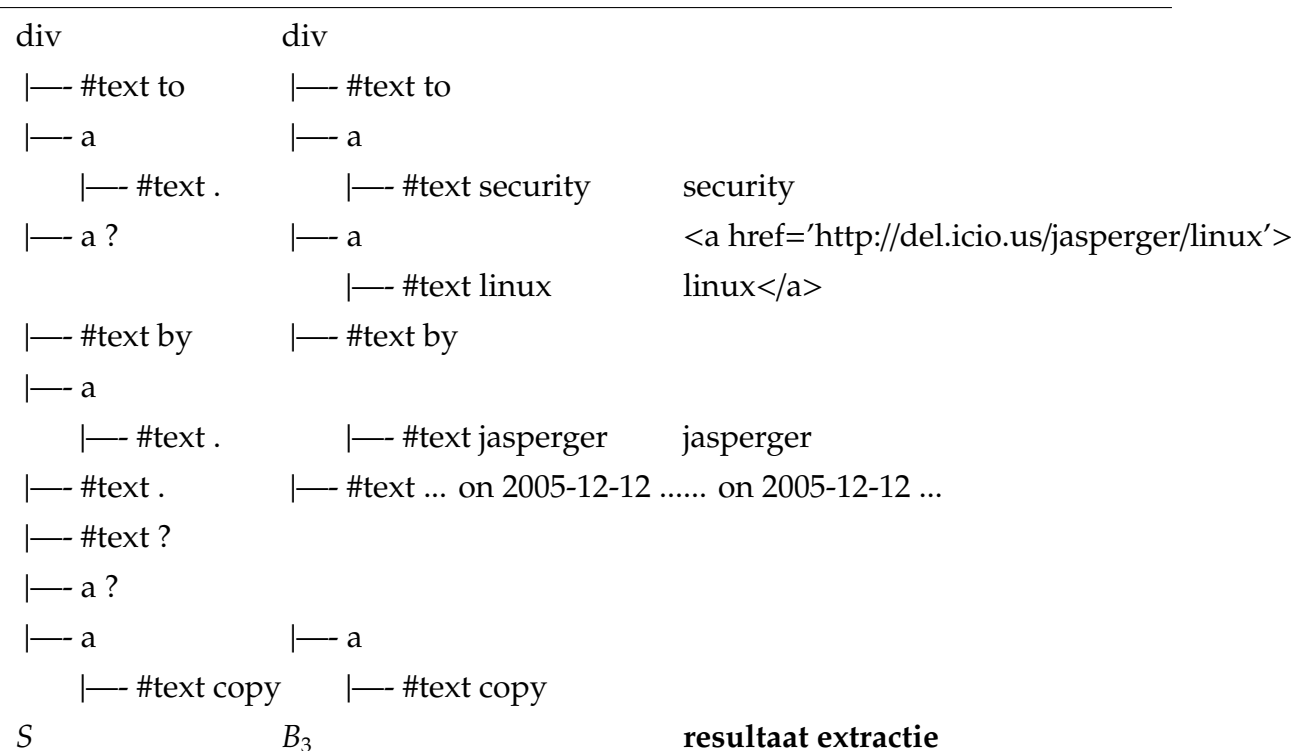
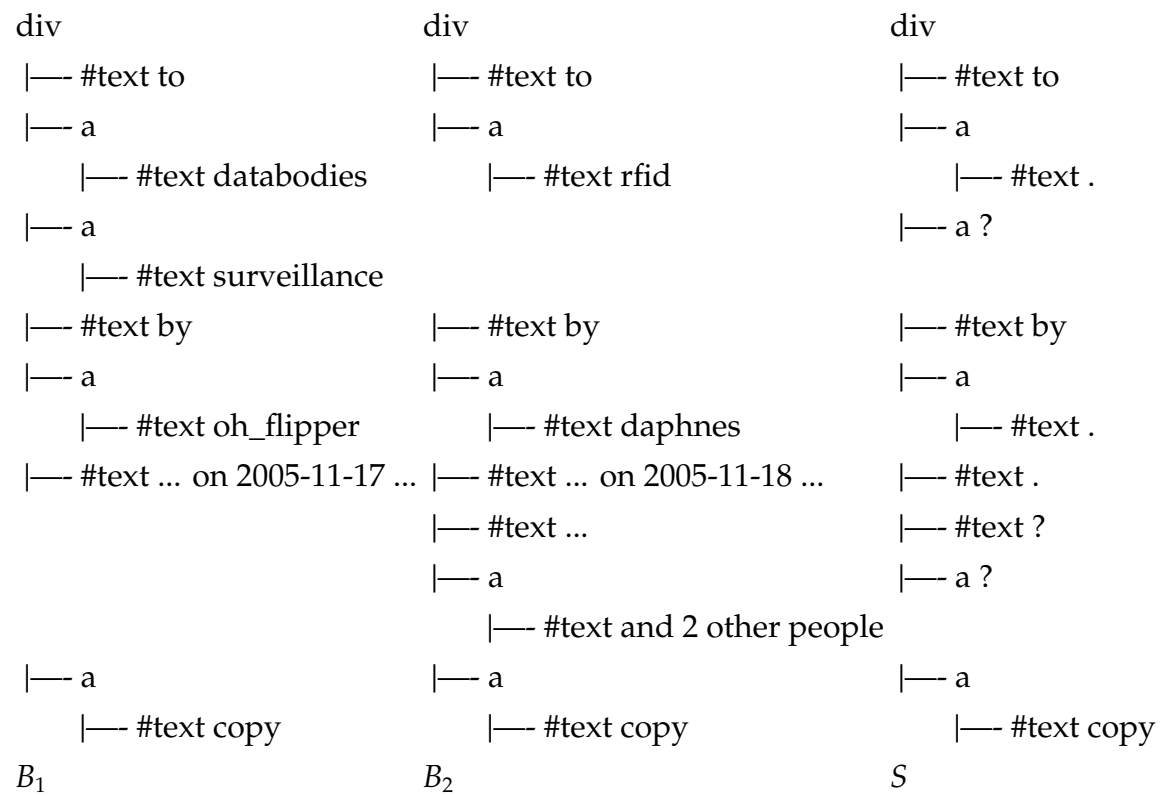
Voor het testen van dit alles hebben we natuurlijk wel een trainings- en een testverzameling nodig. Deze worden in de volgende sectie besproken. Daarna volgt een bespreking van de experimenten.

Als voorbeeld van het hele proces staat in Figuur 6.1 de veralgemening van twee bomen en de toepassing van het resulterende sjabloon op een derde boom.

6.1 Trainings- en testverzamelingen

Deze scriptie is geschreven om te voldoen aan de behoefte om de inhoud van weblogs en nieuws te verkrijgen. Elk artikel van zo een blog of nieuwssite heeft een aparte webpagina. Voor het creëren van een trainingsverzameling voor dit soort sites moet dus alleen maar een aantal artikelen gevonden worden. Om de algoritmes te testen heb ik tien webpagina's van de blogs anjo.blogspot.com, wordpress.justlol.net en informationlab.org verzameld. Voor het nieuws heb ik tien webpagina's van bbc.co.uk en reuters.co.uk gekozen. Mijn algoritme werkt echter niet alleen op de boomrepresentatie van volledige pagina's, maar kan ook leren op deelbomen binnen een pagina, als er verschillende instanties van hetzelfde semantische type op een pagina staan. We duiden de leervoorbeelden dan aan met mijn Firefox-extensie. Hiervoor heb ik een pagina van del.icio.us en google.com (met de resultaten van een willekeurige zoekopdracht) gekozen; beide bevatten tien instanties van hetzelfde semantische type.

Voor deze scriptie worden de sjablonen unsupervised geleerd; het testen van de juistheid van de sjablonen zal echter supervised zijn; deze sjablonen worden getest op gelabelde pagina's. In deze gelabelde pagina's zijn de verschillen tussen een verzameling van pagina's aangeduid. Zodoende is geweten wat de gewenste elementen zijn en kan getest worden of de leeralgoritmes wel de juiste elementen in een pagina als joker hebben aangeduid. Door de gelabelde elementen uit de testverzameling te vergelijken met de elementen die geëxtraheerd worden door het geleerde sjabloon kunnen we kijken hoe goed dat sjabloon is. Wederom heb ik tien pagina's van de genoemde sites verzameld. Voor het labelen heb ik gebruik van de door mij ontwikkelde Firefox-extensie. Deze extensie stelt de gebruiker namelijk ook in staat om bepaalde delen in een webpagina te labelen. Dit gebeurt door een extra attribuut aan het element toe te voegen. Niet alle elementen in een DOM-boom kunnen op deze manier gelabeld worden, omdat niet alle HTML-elementen een attribuut kunnen hebben. Dit is bijvoorbeeld het geval voor de markerings waarmee de titel van een pagina en het commentaar aangeduid worden. Dit is ondervangen door zulke elementen met een speciale tekstuele aanduiding weer te geven in de broncode.



Figuur 6.1: De veralgemening van twee bomen B_1 en B_2 en de toepassing van het resulterende sjabloon S op een derde boom B_3 .

6.2 De vergelijking van de snelheid tussen de beperkte top-down boomwijzigingsafstand en mijn algoritme

In tabel 6.1 staat de vergelijking in snelheid tussen de beperkte top-down boomwijzigingsafstand (*btdbwa*) en mijn benadering. Al deze tests zijn uitgevoerd op een 3GHz processor met 1GB RAM. De metingen zijn uitgedrukt in het gemiddeld aantal seconden benodigde om een afbeelding te leren van twee bomen van één site. Er wordt gemiddeld over de veralgemening van tien keer twee HTML-bomen. De tijd benodigd om het sjabloon te construeren wordt niet meegerekend, aangezien dit sowieso een lineaire tijd in de grootte van de boom heeft. De eerste kolom van tabel 6.1 geeft de gemiddelde grootte van de leervoorbeelden aan; de laatste geeft de verbetering weer van mijn methode ten opzichte van de andere methodes. Deze verbetering is berekend als $btdbwa/benadering$.

Het is duidelijk dat mijn benadering een aanzienlijke tijds winst oplevert, aangezien de verbetering 1.42 tot 15.74 keer sneller is. Het blijkt dat mijn methode vooral veel winst oplevert op grote bomen met een grote vertakingsgraad waar veel elementen met hetzelfde label voorkomen. Dit is ook logisch, aangezien er voor de beperkte top-down boomwijzigingsafstand recursief veel deelbomen met elkaar uitgelijnd moeten worden; bij mijn benadering hoeft dat niet. Het grote verschil tussen de leertijden voor grote bomen met nagenoeg evenveel knopen kan op dezelfde manier verklaard worden; voor bomen die op een bepaald deelniveau veel knopen met hetzelfde label hebben moeten veel meer berekeningen uitgevoerd worden.

6.3 Aantal leervoorbeelden

Reis en Hogue beweerden in hun artikelen dat er telkens maar twee of drie leervoorbeelden nodig zijn om een goed sjabloon te creëren. Voor grote bomen blijkt dit niet het geval te zijn. In tabel 6.2 staat weergegeven hoeveel van de trainingsvoorbeelden uitgelijnd konden worden met het sjabloon dat tot dan toe was geleerd. Voor de kleine bomen, *delicious* en *google*, zijn er al snel sjablonen gevonden die met meer dan de tot dusver geleerde voorbeelden kunnen uitlijnen. Voor de grote bomen duurt dit vaak een stuk langer. Wat erg van belang is is de variatie in de leervoorbeelden. Als in de eerste paar leervoorbeelden alle verschillen in de verzameling aanwezig

site	boom	benadering	btdbwa	verbetering
anjo	220.25	0.20	1.89	9.45
dailykos	613.30	3.85	43.48	11.29
informationlab	493.20	3.18	13.56	4.26
justlol	477.80	21.84	69.30	3.17
bbcnews	637.70	9.60	38.68	4.03
reuters	472.20	3.81	17.23	4.53
del.icio.us	23.10	0.07	0.10	1.43
google	37.70	0.12	0.17	1.42

Tabel 6.1: Vergelijking van de gemiddelde leersnelheid benodigd voor de veralgemening van tien keer twee HTML-bomen, in seconden.

zijn, zijn er weinig leervoorbeelden nodig. Zo kan het sjabloon van delicious al na drie of vier leervoorbeelden met 90% van de leervoorbeelden uitlijnen. Het laatste voorbeeld heeft echter een facultatief element wat in geen enkel van de andere voorbeelden voorkomt. Er is hier dus duidelijk sprake van een plateau tijdens het leren. Wat opvalt is dat mijn benadering vaak sneller een sjabloon vindt dat met een groter deel van de leervoorbeelden uitlijnt, dan het sjabloon geconstrueerd op basis van de wijzigingen die berekend zijn met behulp van de beperkte top-down boomwijzigingsafstand. Ook valt op dat er voor bbcnews nooit een sjabloon gevonden wordt dat met alle bomen uitlijnt. Dit blijkt te komen omdat mijn voorbeelden weliswaar allemaal van de BBC-site afkomstig zijn, maar er verschillende categorieën binnen hun site zijn, zoals wereldnieuws en sport, die niet unificeerbaar zijn.

6.4 Precisie

Precisie is de waarde die aanduidt hoeveel van de geëxtraheerde deelbomen juist zijn. Door de geëxtraheerde deelbomen te vergelijken met de gelabelde deelbomen is dit juist wat gemeten kan worden.

Kushmerick [4] merkte terecht op dat we alleen geïntereiseerd zijn in sjablonen met een precisie van 100 procent. We willen immers een sjabloon dat een perfecte veralgemening is van de voorbeelden. Uit de experimenten blijkt dat de sjablonen over het algemeen prima de verschillen tussen de bomen aanduiden en dus een precisie van 100 procent hebben. Na i leervoorbeelden haalt het extractieproces alle

site	Benadering								
voorbeelden	2	3	4	5	6	7	8	9	10
anjo	30	50	60	60	70	80	90	100	100
dailykos	20	30	40	50	60	70	80	90	100
informationlab	20	30	40	50	60	90	100	100	100
justlol	20	30	40	60	70	70	80	90	100
bbcnews	20	30	40	60	70	70	90	60	60
reuters	20	30	40	60	60	80	90	90	100
del.icio.us	20	90	90	90	90	90	90	90	100
google	30	40	60	60	60	70	80	90	100

site	Reis								
voorbeelden	2	3	4	5	6	7	8	9	10
anjo	20	60	70	70	80	80	90	100	100
dailykos	20	30	40	50	60	70	80	90	100
informationlab	20	30	40	50	60	90	100	100	100
justlol	20	30	50	50	60	70	80	90	100
bbcnews	20	30	30	40	40	40	60	60	60
reuters	20	30	40	60	60	80	90	90	100
del.icio.us	20	60	90	90	90	90	90	90	100
google	30	40	60	60	60	70	80	90	100

Tabel 6.2: Procentuele vergelijking van het aantal leervoorbeelden dat uitlijnt met een sjabloon gecreëerd na het leren op x voorbeelden.

site	geëxtraheerd	te veel	correct	gelabeld	niet gevonden
anjo	16.13	1.38	14.75	14.75	0
dailykos	47.33	1	47	47	0
informationlab	19	2.4	16.6	16.6	0
justlol	47.86	1.14	46.71	46.71	0
bbcnews	n/a	n/a	n/a	n/a	n/a
reuters	34	2	32	32	0
del.icio.us	6.3	1.1	5.4	6.2	1
google	8.4	1	7.4	7.4	0

Tabel 6.3: Het gemiddelde aantal geëxtraheerde deelbomen voor sjablonen gecreëerd met de beperkte top-down boomwijzigingsafstand.

gelabelde deelbomen uit de $B_{1...i}$ geleerde bomen. In de tabellen 6.3 en 6.4 staat een overzicht van het gemiddeld aantal geëxtraheerde deelbomen voor respectievelijk de sjablonen gecreëerd met behulp van de beperkte top-down boomwijzigingsafstand en die gecreëerd met mijn benadering. In de eerste kolom staat beschreven hoeveel deelbomen er met behulp van het uiteindelijke sjabloon uit de trainingsvoorbeelden werden geëxtraheerd. Al deze voorbeelden zijn gelabeld en we weten dan ook precies hoeveel en welke knopen we willen dat het sjabloon extraheert. In de tweede kolom staat hoe veel deelbomen er foutief zijn geëxtraheerd, in de derde kolom hoeveel er correct zijn geëxtraheerd. De vierde kolom laat zien hoeveel deelbomen er door mij gelabeld zijn. De laatste kolom toont hoeveel van die gelabelde knopen er uiteindelijk niet zijn gevonden. Aangezien de trainingsverzameling van bbcnews niet unificeerbaar is en er geen goed sjabloon is geleerd, is hier niet op getest.

Wat meteen opvalt is dat er voor beide methodes vaak te veel geëxtraheerd wordt. Nadere analyse liet zien dat dit in het geval van anjo, dailykos, informationlab, justlol en google telkens kwam omdat ik een #comment of een link niet had gelabeld. In het geval van mijn benadering werd er voor justlol ook nog een #comment meer geëxtraheerd dan in de beperkte top-down variant. Dit wordt later verder besproken. Voor de rest hadden al deze wrappers voor beide methodes dus een precisie van 100 procent aangezien alle gelabelde bomen geëxtraheerd zijn en niets te veel geëxtraheerd is.

Voor delicious, reuters en die ene extra #comment in justlol is er iets anders aan de hand. Een diepere analyse van delicious laat zien wat. Voor delicious merken

site	geëxtraheerd	te veel	correct	gelabeld	niet gevonden
anjo	16.13	1.38	14.75	14.75	0
dailykos	47.33	1	47	47	0
informationlab	19	2.4	16.6	16.6	0
justlol	48.57	1.86	46.71	46.71	0
bbcnews	n/a	n/a	n/a	n/a	n/a
reuters	35.75	32	3.75	32	28.25
del.icio.us	8.6	2.8	5.8	6.2	0.4
google	8.4	1	7.4	7.4	0

Tabel 6.4: Het gemiddelde aantal geëxtraheerde deelbomen voor sjablonen gecreëerd met mijn benadering van de beperkte top-down boomwijzigingsafstand.

we dat de resultaten van de beperkte top-down variant gemiddeld slechter zijn dan die van de beperkte boomwijzigingsafstand. In het geval van de beperkte top-down variant blijkt dat alle gewenste knopen wel worden geëxtraheerd, maar dat ook telkens drie tekstknopen te veel worden geëxtraheerd. Herinner de bespreking van Figuur 5.2 en sjabloon S_1 . Daar werd aangetoond dat bij het iteratief opbouwen van het sjabloon met behulp van de beperkte top-down afbeelding het goedkoper is om een tekstknoop in te voegen of te verwijderen dan om zijn burens, die grotere deelbomen zijn, in te voegen of te verwijderen. Dit is precies wat er hier aan de hand is. Ik heb geprobeerd dit te verbeteren, door in mijn benadering gelijke elementen zoveel mogelijk bij elkaar te houden en verwijderingen of invoeringen van grote knopen toe te staan, als zij vergeleken worden met een element met hetzelfde label. Daardoor wordt de optionele 'a' uit B_2 ingevoegd in plaats van uitgelijnd met het verkeerde element. Uit de resultaten van mijn benadering voor delicious blijkt deze aanpak ook te werken, maar struikelen we bij het laatste voorbeeld over het probleem besproken in Figuur 5.3 voor sjabloon S_2 . Hierdoor wordt een ouderknoop van de tot dusverre structureel onderscheidende elementen als facultatief gemerkt. De 0.4 in de kolom niet gevonden duidt dan ook op de 4 knopen die bij de veralgemening met de laatste pagina opgeslokt zijn. Dit betekent dat we nog wel alles extraheren wat we willen, maar dat er ook knopen bij zitten die structureel bepalend en dus niet gewenst in de extractie zijn. Dit laat zien dat de volgorde bij het leren van belang kan zijn.

Ook bij reuters komt dit tweede probleem voor. Doordat een facultatief element

met een structureel element met hetzelfde label wordt uitgelijnd, wordt dat structureel element ook facultatief gemaakt. Dit is het nadeel bij niet recuseren. Als er vele elementen met hetzelfde label naast elkaar staan weten we immers niet welk element hetzelfde moet blijven. De resultaten tonen dat dit gelukkig niet vaak voor komt.

De extra `#comment` in `justlol` blijkt ook een gevolg van mijn benadering te zijn. Omdat het niets kost om aangrenzende elementen met hetzelfde label te verwijderen of in te voegen, gebeuren er daar soms te veel van. Dit leidt dan tot de extractie van een extra `#comment`, die niet verschilt tussen de bomen.

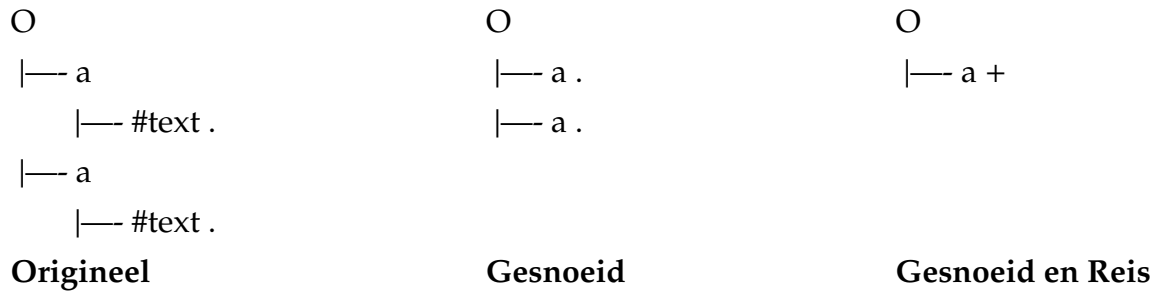
Reis beweerde in zijn artikel dat `single` en `plus` de gewenste elementen representeren en `optional` en `kleene` de facultatieve elementen. Dit blijkt echter vaak niet juist te zijn. Reclame kan bijvoorbeeld op elke pagina op dezelfde plaats staan, maar de tekst kan telkens anders zijn. De joker hiervoor is echter wel een `single`. Doordat dit soort 'foute' singles ook gebruikt wordt in een nabewerkingsstap die de joker opwaardeert tot een `plus`, bevat ook de `plus` niet altijd de gewenste elementen. Bij de facultatieve elementen `optional` en `kleene` is me opgevallen dat zij niet altijd enkel overbodige informatie representeren. Soms staat iets maar in één boom, wat aangeduid wordt als facultatief, maar is het wel interessant.

Niet alleen de verschillen tussen de bomen in de leervoorbeelden zijn interessant. Als we bijvoorbeeld alleen maar artikelen die op dezelfde dag geschreven zijn als leervoorbeelden opgeven, wordt de datum niet als joker aangeduid omdat deze niet verschilt tussen de bomen. De leervoorbeelden geven dus een afwijking aan het sjabloon als er niet genoeg variatie in de bomen zit. Dit probleem is onvermijdelijk bij *unsupervised machine learning*; de trainingsgegevens moet representatief genoeg zijn — er moet genoeg variatie zijn — om een goede wrapper te kunnen leren.

Er kan dus gesteld worden dat de sjablonen de data wel goed representeren, maar dat er ook een aantal ongewenste deelbomen tussenzit. Ook kunnen gewenste deelbomen, zoals de datum, ontbreken als die in de leervoorbeelden telkens hetzelfde zijn.

6.5 Nabewerkingen

Een nabewerkingsstap wordt toegepast als het sjabloon geleerd is. Ik heb twee soorten nabewerkingsstappen getest: die van Reis, en het snoeien, zoals beschreven op pagina 29. Hogues methode heb ik niet toegepast, omdat die veel te duur is voor



Figuur 6.2: Het effect van eerst snoeien en dan Reis' nabewerkingsstap.

de grote bomen uit onze context. De bedoeling van een nabewerkingsstap is om bijvoorbeeld lijsten met een variabele lengte te kunnen extraheren; dat blijkt ook te kunnen.

De beste resultaten worden over het algemeen verkregen door eerst het sjabloon te snoeien en daarna de nabewerking voorgesteld door Reis toe te passen. Dit komt omdat zonder snoeien jokers vaak geen burens zijn omdat ze te diep in een deelboom zitten. Ter illustratie bekijken we Figuur 6.2 waar twee buurknopen een link zijn. Een link heeft vaak maar één kind, een tekstknoop. Als deze tekstknopen een joker-aanduiding hebben zullen ze niet door Reis' methode gecombineerd worden, omdat ze niet dezelfde ouder hebben. Als er gesnoeid wordt zullen de jokers verplaatst worden van de tekstknoop naar de link. Hierna zijn er twee jokerburens, de gesnoei-de links. In dit geval kan Reis' methode deze wel omzetten in een gecombineerde jokerknoop, in dit geval een plus.

Hoofdstuk 7

Gerelateerd werk en toepassingen

7.1 Gerelateerd werk

Mijn benadering van sjablooninductie is een onderdeel van het grotere onderzoeksgebied van de informatie-extractie. De laatste jaren is dit gebied uitgebreid onderzocht, zeker als het gaat om documenten op het Wereldwijde Web (WWW). Informatie-extractie omvat het geautomatiseerd binnenhalen van gegevens van gestructureerde en ongestructureerde documenten. Verschillende manieren zijn uitgeprobeerd; de mate van succes is uiteenlopend.

Het gedeelte van de informatie-extractie dat zich bezighoudt met documenten op het Wereldwijde Web wordt, zoals eerder gezegd, wrapper-inductie genoemd. Dit kan supervised of unsupervised gebeuren op de HTML als platte tekst of als een boomstructuur.

Voorbeelden van supervised wrapperinductiesystemen die op platte tekst werken zijn WHISK [12], STALKER [13], en Kushmerick [14]. Deze systemen delen een aantal eigenschappen: 1) De wrapper-generator gebruikt extra informatie zoals een verzameling van gelabelde voorbeelden. De wrapper wordt afgeleid door het bekijken van deze voorbeelden waarna ze worden veralgemeend. 2) Meestal heeft het wrapperinductiesysteem voorkennis over de indeling van een pagina, bijvoorbeeld over het schema van de gegevens in de pagina; de meeste nemen aan dat de doelpagina's een verzameling van wederkerende gegevens hebben. In andere gevallen kan het systeem ook geneste gegevens aan, maar moet het weten welke attributen geëxtraheerd moeten worden en hoe ze genest zijn. 3) Deze systemen genereren een wrapper door één pagina per keer te bekijken.

Een andere manier om een wrapper voor een HTML-pagina te leren kan het

unsupervised afleiden van een, gewoonlijk reguliere, grammatica voor de HTML-code zijn. Grammatica-afleiding is een welbekend en goed bestudeerd domein [15]. De afleiding van een reguliere grammatica is echter moeilijk: het is bekend uit Golds werk [16] dat reguliere grammatica's niet correct geïdentificeerd kunnen worden op basis van enkel positieve voorbeelden. In het geval van positieve en negatieve voorbeelden bestaat trouwens geen efficiënt leeralgoritme dat een 'minimum state deterministic finite state automaton' kan vinden dat consistent is met een willekeurige verzameling voorbeelden [17]. Daarom hebben algoritmes die dit soort afleidingen gebruiken extra informatie nodig, zoals een verzameling gelabelde voorbeelden of de antwoorden van een deskundige op vragen van de leerder.

ROADRUNNER [18] implementeert een volledig geautomatiseerd, unsupervised, wrappergeneratieproces op dynamisch gegenereerde pagina's zonder de gegevens vooraf te labelen. Het vergelijkt pagina's van een query-verzameling en probeert een (union free) reguliere expressie te vinden die alle pagina's uitlijnt. Dit gebeurt door over tokens uit twee pagina's te itereren en te zoeken naar foutieve uitlijningen. Deze foutieve uitlijningen worden dan gebruikt om de facultatieve delen van de pagina, de iteratief gegenereerde velden en recursieve gegevensstructuren te vinden. Ook heeft ROADRUNNER geen voorkennis nodig.

ROADRUNNER lijkt qua mogelijkheden zeer op mijn systeem en is snel en efficiënt, maar slaagt er soms niet in een wrapper te genereren. Dit gebeurt als een patroon de vorm heeft van een context-vrije taal, of als het een unie van reguliere talen betreft. Aangezien de boomwijzigingsafstand niet probeert te zoeken naar een reguliere taal in platte tekst komt dit probleem bij mij niet voor. Mijn systeem werkt dan ook op de onderliggende boomstructuur van HTML en is niet beperkt tot het zoeken naar een reguliere taal. Eerdere pogingen om interessante gegevens van webpagina's te verkrijgen zagen deze onderliggende structuur vaak over het hoofd; ze keken alleen naar de HTML-markeringen die in de onmiddellijke nabijheid van de gewenste informatie liggen.

Sommige leermodellen delen gegevens echter in door te veronderstellen dat elementen die dicht bij elkaar liggen in een hiërarchie ook op gelijkaardige wijze ingedeeld moeten worden [19]. De documentbomen van een bestandssysteem of van HTML-pagina's zijn bijvoorbeeld handig om vast te stellen of documenten gerelateerd zijn. Als de meeste documenten in een bepaalde directory of deelboom ingedeeld zijn, is het zeer waarschijnlijk dat nieuwe documenten die in nabijgelegen deelbomen voorkomen, op gelijkaardig wijze ingedeeld kunnen worden. Deze

modellen gebruiken echter alleen maar de structuur van de HTML-bomen en niet ook de labels van de knopen en eventueel de inhoud van structureel belangrijke tekst.

Algoritmes die boom patronen vergelijken (*tree pattern matching algorithms*) zijn uitgebreid bestudeerd in gebieden zoals de compiler optimalisatie en de moleculaire biologie [20, 21, 22]. Traditioneel heeft men zich gericht op manuele zoekopdrachten van gebruikers. Dit veronderstelt zowel een uitgebreide kennis van de semantiek van patroonafbeelding als van de onderliggende gegevensstructuur. Meestal richt men zich op de vergelijking van structuren in niet-geordende bomen, met of zonder jokers.

Een aanverwant probleem is het deelboomisomorfisme [23, 24] dat poogt gelijkvormige structuren in bomen te vinden. Over het algemeen benaderen deze algoritmes de structuur van een gegeven boom zonder naar de labels van een knoop of naar de volgorde van de afstammelingen te kijken. Het gaat hierbij bijna altijd om specialisaties van de moeilijker opdracht om graafisomorfismen te vinden.

Mijn benadering van de beperkte top-down boomwijzigingsafstand heeft de complexiteitsproblemen van voorgaande algoritmes vermeden door op te merken dat een verzameling van automatisch gegenereerde HTML-pagina's van eenzelfde site, top-down is en dat de volgorde en de labels van de bomen waarmee gewerkt wordt bewaard moeten blijven.

7.2 Toepassingen

De methodes beschreven in deze scriptie hebben allerlei toepassingen. Op de eerste plaats komt natuurlijk de issuescraper met het oog waarop dit onderzoek begonnen is. Mijn leeralgoritme kan gebruikt worden om een sjabloon te maken waarmee de resultaten van zoekmachines binnengehaald kunnen worden voor het vergaren van artikelen. Voor elke verkregen bron wordt dan bekeken of er al een sjabloon voor bestaat en of het nog geldig is (uitlijnt). Zoniet kunnen we het leeralgoritme een sjabloon laten creëren door een gebruiker of 'superuser'. Mijn implementatie kan als module in de issuescraper opgenomen worden. We zijn ook bezig met een scriptje dat automatisch het archief van een site opzoekt en pagina's hiervan als leervoorbeeld neemt. Zodoende wordt het proces nog meer geautomatiseerd. In het volgende hoofdstuk staat een verdere bespreking van het gebruik sjablonen in de issuescraper.

Een andere toepassing is het creëren van rss-feeds of XML-documenten. De extractie en omzetting naar een rss-feed met behulp van gelabelde sjablonen is een trivialiteit.

In mijn inleiding heb ik een korte schets gegeven van de interpretaties van links. Afgezien van uiteenlopende interpretaties verandert ook de kwaliteit van het linken. Een voorbeeld hiervan is de opkomst van het zogenaamde mobtaggen¹. Mobtags zijn labels die door een gebruiker aan een link, plaatje of artikel worden toegevoegd om ze te classificeren of een naam te geven. Deze manier van taggen kan beschouwd worden als de ontwikkeling van een ontologie. De opkomst van taggingsystemen zoals del.icio.us, flickr.com, digg.com, de integratie van del.icio.us in yahoo enzovoort noemt men een belangrijk onderdeel van Web 2.0². Met behulp van mijn Firefox-extensie en het leren van sjablonen kunnen bepaalde delen van een webpagina naar willekeur getagd worden voor later gebruik. Op dit moment gebeurt het leren via een programma op een server, maar dit zou omgezet kunnen worden naar client-side javascript. De templates kunnen dan opgeslagen worden in cookies. Op deze manier kunnen dezelfde soort gegevens op dezelfde manier automatisch getagd worden.

Bij het uitwerken van mijn methodes heb ik er ook zorg voor gedragen dat ze goed zouden werken op de herhalende resultaten van zoekmachines. Door sjablonen van verscheidene zoekmachines te maken is het zeer eenvoudig om metazoekmachines te ontwikkelen die de resultaten van verschillende sites combineren. Tot nu toe moet ik, als er meerdere semantische instanties op een pagina staan, een aantal van die instanties aanduiden. Een volgende uitdaging is om mijn leerscript de instanties automatisch te laten vinden. Een mogelijke manier is om van twee pagina's met evenveel resultaten een sjabloon te leren. In dit sjabloon kan dan het langste gemeenschappelijke voorouderpad tot de verschillen gezocht worden. Dat pad kan dan gebruikt worden als een relatieve XPath query voor het vinden van de individuele semantische instanties. Een moeilijkheid hierbij is wel dat een instantie

¹Mobtagging is een term die in het leven geroepen is door mediamatic.net. Ik vind dit een goede term omdat deze aangeeft dat het taggen gebeurt door een ongeorganiseerde groep mensen.

²Web 2.0 is een term die vaak gebruikt wordt voor de overgang van het Wereldwijde Web van een verzameling van websites naar een compleet computingplatform dat webapplicaties aan eindgebruikers aanbiedt. Uiteindelijk wordt verwacht dat Web 2.0 desktop computing zal vervangen voor allerlei doeleinden. Het is een sociaal fenomeen dat de creatie en distributie van de inhoud van het Web omvat. Dit wordt gekenmerkt door communicatie, decentralisatie, vrijheid om te delen en te hergebruiken en 'de markt als conversatie'.

over naburige deelbomen verspreid kan zijn. Dit is geen handige structuur omdat de boomwijzigingsafstand niet gedefinieerd is op bossen. Wellicht kan mijn benadering hier beter mee overweg.

Apple heeft sinds OsX 10.4 widgets geïntroduceerd. Dit zijn kleine programmaatjes die informatie zoals het weer en beursberichten van internet ophalen en weergeven op je desktop. Om deze informatie te verkrijgen moet je natuurlijk een sjabloon hebben. Raad maar hoe dit gemaakt kan worden ...

Aangezien de sjablonen automatisch geconstrueerd worden en de gebruikers-interface heel simpel is - klikken op je rechtermuisknop - is het nu mogelijk voor *Jan met de pet* om wrappers te maken en informatie automatisch tot bij hem te laten komen.

Met de resultaten van mijn onderzoek en de implementaties ervan is de overgang naar het semantisch web en Web 2.0 gemakkelijker te realiseren. Het is immers een illusie te denken dat alle sitebouwers zullen overschakelen naar XML. Ook kunnen gegevens op verschillende manieren getagd, beschreven of in een ontologie opgenomen worden. De interpretatie van gegevens is immers afhankelijk van context en domein. Door gebruikers zelf gegevens te laten verkrijgen en beschrijven kunnen zij deze op een voor hen relevante manier hergebruiken.

Hoofdstuk 8

Conclusie en discussie

Bij wrapperinductie kan voor het leren van een sjabloon gebruik gemaakt worden van boomwijzigingsafstanden waarbij de wijzigingen beperkt worden tot de blaadjes. Dit kan omdat in een verzameling automatisch gegenereerde HTML-pagina's van dezelfde site, de verschillen tussen de pagina's in of vlakbij de bladeren van de boomrepresentatie van die pagina's zitten. Het overgrote deel van deze verschillen geeft aan waar de interessante gegevens zich bevinden. Het geleerde sjabloon, een veralgemening van de voorbeeldpagina's, duidt de verschillen tussen de bomen in de trainingsverzameling aan en kan gebruikt worden om de interessante gegevens in andere pagina's van dezelfde site te herkennen.

In deze scriptie heb ik laten zien dat als de verschillen tussen de bomen als jokers gerepresenteerd worden hiermee rekening gehouden moet worden bij de prijs van de wijzigingen. Ik heb aangetoond dat de reeds bekende boomwijzigingsafstanden voor wrapperinductie vaak vele onnodige recursies uitvoeren om de verschillen tussen twee bomen te achterhalen. Vervolgens heb ik een nieuw algoritme voorgesteld dat deze overbodige bewerkingen achterwege laat door de boomwijzigingsafstand te benaderen. Dit leidt tot een aanzienlijke verbetering van de snelheid in het leren van een sjabloon. Opvallend is dat de plaats en het label van een element, in verhouding tot zijn burens, een goede aanduiding geeft of iets gewijzigd moet worden. Ik heb aangetoond dat mijn benadering meestal een even goed sjabloon oplevert als die verkregen met de beperkte top-down boomwijzigingsafstand. De gevallen waar dit niet zo is zijn uitgelegd; en verwacht wordt dat door een nadere bestudering van de mogelijke structuur en opbouw van HTML-markeringen ook hier een oplossing voor gevonden kan worden.

De mij bekende applicaties die de boomwijzigingsafstand voor wrapperinduc-

tie gebruiken zijn beschreven door Reis [5] en Hogue [6]. Beiden passen een nabewerkingsstap toe om enerzijds de sjablonen compacter te maken en anderzijds aangrenzende deelbomen met een variabele hoeveelheid uit te lijnen. Hogues manier is erg prijzig maar hij past deze alleen toe op kleine deelbomen. Voor blogs en nieuws, waar de artikelen een hele pagina beslaan, is ze te duur. Reis' manier is veel goedkoper en blijkt regelmatig goed te werken. Ik heb nog een extra nabewerkingsstap geïmplementeerd die jokerknopen naar de ouder verschuift als al zijn kinderen jokers zijn. Dit blijkt een zinvolle stap in combinatie met die van Reis. De nabewerkingsstappen zijn essentieel, omdat ze ook de interpretatie van de jokerknopen veranderen. Dit laat ons toe om bijvoorbeeld lijsten met een variabele lengte te extraheren; het sjabloon wordt daardoor expressiever en de afwijking van het sjabloon op de testverzameling wordt geminimaliseerd.

Het vinden van verschillen tussen bomen leidt niet altijd tot een sjabloon dat voor alle gewenste knopen jokers heeft. Dit is bijvoorbeeld het geval als telkens dezelfde datum in de leervoorbeelden voorkomt. Aangezien het niet altijd mogelijk is om een trainingsverzameling te maken waarin de gewenste gegevens verschillen, heb ik een manuele oplossing gecreëerd: na het leren kan een gebruiker het sjabloon controleren en met behulp van de Firefox-extensie extra jokers, die een knoop of deelboom vervangen, aan het sjabloon toevoegen. Hiervoor kan de gebruiker een sjabloon voor een bepaalde site laden en wordt visueel aangeduid welke delen van de pagina geëxtraheerd worden. Als de gebruiker ziet dat er iets ontbreekt kan hij dat via een contextmenu aanduiden en in het sjabloon opslaan. Met behulp van de extensie kunnen ook jokers, zoals reclame, verwijderd worden.

De methodes beschreven in deze scriptie vertellen wel waar de interessante gegevens staan, maar niet wat die gegevens precies inhouden. Deze kennis is gewenst als we iets met die gegevens willen doen. Hiertoe moet een labelstap gebruikt worden. Als we ons domein goed genoeg beheersen kunnen we gebruiken maken van een heuristiek. Als het bijvoorbeeld om het domein van een nieuwsartikel gaat, weten we dat de titel bovenaan de pagina staat en dat de auteur en de datum in de buurt daarvan staan. De datum bestaat voornamelijk uit cijfers en de naam van de auteur is over het algemeen korter dan de titel. Soms kan ook gebruik gemaakt worden van de klassenamen van bepaalde knopen. Het is ook mogelijk om de tekst net voor of net na een knoop te bekijken. Dit alles heb ik niet geïmplementeerd, maar mijn Firefox-extensie geeft een gebruiker de mogelijkheid om jokers te labelen.

Het doel van het onderzoek in deze scriptie is in ieder geval bereikt: er kunnen nu

zeer snel goede sjablonen gecreëerd worden die de feitelijke artikelen van weblogs en nieuwssites kunnen binnenhalen. Het enige wat voorlopig manueel moet gebeuren is het aangeven van de leervoorbeelden en het labelen van het sjabloon. Bij het kiezen van de leervoorbeelden moet voorlopig wel rekening gehouden worden dat ook binnen één site, zoals die van de BBC, verschillende sjablonen gebruikt kunnen worden voor verschillende categorieën van artikelen.

Soms veranderen websites van lay-out en dus van sjabloon. Dit kan eenvoudig gedetecteerd worden: de extractie moet immers foutloos verlopen. Zoniet moet een nieuw sjabloon voor de website geleerd worden.

Voorlopig onopgelost is het probleem dat de boomwijzigingsafstanden enkel op knopen werken. Deze knopen geven de opmaak van een HTML-document aan. Dit geeft soms problemen als we maar een deel van een doorlopende tekst, die met één tekstknoop wordt aangeduid, willen extraheren. Dit komt bijvoorbeeld voor als er '29 resultaten' op een pagina staat. Het is denkbaar dat voor sommige extractietaken alleen de '29' gewenst is. Met de in deze scriptie beschreven methodes kan dit niet bereikt worden.

In deze scriptie is aangetoond dat een grote verscheidenheid aan sjablonen, of wrappers, automatisch geleerd kunnen worden en dat deze praktisch inzetbaar zijn voor een grote verscheidenheid aan taken. Ook al zullen er voor de issuescraper nog wat manuele stappen moeten gebeuren, zoals het controleren en het labelen van de sjablonen, is het nu erg makkelijk om de benodigde sjablonen te creëren. En zoals John Dewey al zei: "Elk waarachtig leerproces, indien geslaagd, leidt ertoe dat je weet hoe iets beter wordt dan vroeger."

Bijlage A

Woordenlijst

Omdat er heel weinig Nederlandse teksten bestaan over kunstmatige intelligentie en informatica liggen niet alle Nederlandse begrippen in deze tekst voor de hand. Daarom hier een woordenlijst. Vertalingen ander andere gevonden via [25, 26, 27, 28].

Nederlands	Engels
afbeelding	mapping
afvlakken	to flatten
afweging	trade-off
afwijking	bias
beperkt	restricted
beperkte top-down afbeelding	restricted top-down mapping
bewerking	operation
bij benadering	approximately
boom	tree
boomwijzigingsafstand	tree-edit-distance
deterministische eindige toestandsautomaat	deterministic finite state automaton
disjuncte samenvoeging	disjoint union
doorkruising	traversal
dynamisch programmeren	dynamic programming
extensie	extension
facultatief	optional
gegevensbank	database
genest	nested

Nederlands	Engels
graaf	graph
gulzig	greedy
herhaald uitvoeren	to iterate
inductie	induction
isomorfisme	isomorphism
in elkaar schuiven	to collapse
itereren	to iterate
joker	wildcard
knoop	node
markering	mark-up
markeringstaal	mark-up language
nauwkeurigheid	precision
nesten	to nest
niet deterministische eindige toestandsautomaat	non deterministic finite state automaton
oorsprong	root
ouder	parent
pad	path
precisie	precision
pre-orde	preorder
postorde	postorder
robuust	robust
sjabloon	template
toestand	state
toestandsstructuur	state structure
uitlijnen	to match
vervangen (als wijziging)	to change
vervangen	to substitute
verwant	sibling
verzameling	set
volledigheid	completeness
voorvader	ancestor
voorvoegsel	prefix
wandeling	walk
wijzigingsafstand	edit-distance
wortel	root
zoekopdracht	query
zoektaal	query language

Bijlage B

Uitleg voor digibeten

Ook mijn Opa weet graag waar ik mee bezig ben. Daarom speciaal voor hem, en voor de anderen die niet zo onderlegd zijn in de techniek van nieuwe media, deze bijlage.

Op het internet staat een gigantische hoeveelheid interessante gegevens. Al deze informatie staat echter verspreid over onnoemelijk veel verschillende sites. Het zou handig zijn als een computerprogramma deze informatie voor ons zou opzoeken en groeperen. Helaas bestaat er geen uniforme manier om deze interessante gegevens met software te benaderen. We kunnen niet zomaar zeggen 'ik wil dit soort informatie van deze site'. Elke site heeft namelijk zijn eigen opmaak en structuur en is ontworpen voor de menselijke lezer, maar niet voor de benadering door een programma. Rond de interessante gegevens staat vaak ook overbodige informatie.

Kenmerkend voor veel van de interessante gegevens op het web is dat de structuren, waarin de interessante informatie vervat zit, zichzelf binnen een website herhalen. Deze structuur is de onderliggende HTML, wat een taal is voor de opmaak van documenten op het internet. HTML zie je niet als je een webpagina bekijkt, maar wordt geïnterpreteerd door je browser waardoor die weet hoe hij het document moet weergeven.

Een webpagina wordt vaak automatisch gegenereerd, waarbij gegevens uit een gegevensbank gehaald en in sjablonen gestopt worden. Deze sjablonen zijn kleine stukjes HTML waar een rij gegevens uit de gegevensbank ingestopt wordt. De combinatie van een aantal gevulde sjablonen vormt een webpagina. Anders verwoord: een webpagina bestaat vaak uit een combinatie van een klein aantal HTML-sjablonen die telkens met andere gegevens gevuld zijn. Zo heeft elk artikel op een nieuwssite hetzelfde sjabloon. Ook elk resultaat van een zoekactie in bijvoorbeeld Google heeft

hetzelfde stukje HTML als markering. Helaas kennen wij als gebruiker van de site het originele sjabloon niet, waardoor we niet alleen de interessante gegevens zien maar ook alle opmaakcodes er rond. We zullen dus zelf een sjabloon moeten afleiden willen we alleen de interessante gegevens verkrijgen.

Als we nu de broncode (de onderliggende HTML) van verschillende artikelen vergelijken stellen we vast dat er veel overeenkomsten zijn. De gegevens verschillen, maar de code er rond is gelijkaardig. Als we deze gelijkaardige code nu zelf veralgemenen tot een sjabloon kunnen we dit vergelijken met een artikel. Zo kunnen we de interessante gegevens van de code onderscheiden.

In deze scriptie onderzoek ik hoe we een stukje software zulke sjablonen automatisch kunnen laten 'aanleren'. Hiertoe baseer ik me op twee artikelen [6, 5]. Ik heb onderzocht hoe deze methodes werken en vastgesteld dat ze nog niet optimaal zijn en ik leg uit hoe de methodes verbeterd kunnen worden. Ook bekijk ik hoe de sjablonen zo goed mogelijk gemaakt kunnen worden, zodat we ons op de interessante gegevens kunnen concentreren.

Bijlage C

English abstract

This paper discusses how the tree-edit-distance may be used for the problem of wrapper induction. The tree-edit-distance is used to find a mapping with minimal cost between the tree representation of HTML pages. With this mapping a template may be constructed in which only the elements common to the trees are kept. The parts specific to each tree are represented as wildcards. The template will thus be the most specific generalization of the pages and may be used for recognizing other pages of the same semantic type. By using the template for extraction on similar pages the instance specific information may be retrieved.

This paper shows that the domain of automatically generated HTML pages contains a number of characteristics with which the tree-edit-distance may be approached and calculated faster. A number of post processing steps are considered to make the templates more condensed en protect them from overfit. It is found that pruning always gives good results. ning always gives good results.

Bronvermelding

- [1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):35, May 2001.
- [2] N. Marres and R. Rogers. To trace or to rub. 1999.
- [3] R. Rogers and N. Marres. Landscaping climate change: A mapping technique for understanding science & technology debates on the world wide web. *Public Understanding of Science*, 2(9):141–163, 2000.
- [4] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper induction for information extraction. *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, 1997.
- [5] D. de Castro Reis, P. Golgher, A. Laender, and A. da Silva. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th International Conference on the World Wide Web*, pages 502–511. WWW2004, May 2004.
- [6] A. Hogue. Tree pattern inference and matching for wrapper induction on the world wide web. Master's thesis, MIT, june 2004.
- [7] *tidy*. <http://tidy.sourceforge.net>.
- [8] W. Chen. New algorithm for ordered tree-to-tree correction problems. *Journal of Algorithms*, 40:135–158, 2001.
- [9] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 3(42):133–139, 1992.
- [10] K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, July 1979.
- [11] C. L. Lu. A tree-matching algorithm based on node splitting and merging. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(2):249–256, 1984.

- [12] S. Soderland. Learning information extraction rules for semistructured and free text. *Machine Learning*, 34:1–3, 1999.
- [13] I. Muslea, S.Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents (AA-99)*, 1999.
- [14] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000.
- [15] L. Pitt. Inductive inference, dfas and computational complexity. In *Analogical and Inductive Inference, Lecture Notes in AI 397*, Berlin, 1989. Springer-Verlag.
- [16] E. M. Gold. Language identification in the limit. *Information and control*, 10(5), 1967.
- [17] E. M. Gold. Complexity of automaton identification from given data. *Information and control*, 37(3), 1978.
- [18] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th VLDB Conference*, Rome, Italy, 2001.
- [19] L. Shih and D. Karger. Learning classes correlated to hierarchy. Technical report, MIT AI Lab, 2001.
- [20] E. D. Demaine, S. Mozes, B. Rossman, and O Weimann. An $o(n^3)$ -time algorithm for tree edit distance. *MIT Computer Science and Artificial Intelligence Laboratory*, 2004.
- [21] M. Dubiner, J. Galil, and E. Magen. Faster tree pattern matching. *ACM*, 41(2):205 – 213, March 1994.
- [22] C. M. Hoffmann and M. J. O’ Donnel. Pattern matching in trees. *ACM*, 29(1), January 1982.
- [23] J. Hopcroft and R. Tarjan. *Isomorphism of Planar Graphs*, pages 131–152. Plenum Press, 1972.
- [24] D. Matula. An algorithm for subtree identification. *SIAM Rev*, 10:273–274, 1968.

- [25] *Woordenboek voor computers, internet en telecommunicatie*. Academic Service Schoonhoven, 2003-2004.
- [26] *Woordenboek Informatica en Telecommunicatie*. Sybex, 6 edition, 2000.
- [27] *Woordenboek informatietechnologie Nederlands-Engels-Duits*. Kluwer, 1998.
- [28] *2400 x liever Nederlands*. <http://snn.vvb.org/woordenlijst.htm>.