Faculteit Wetenschappen

2de Master in de Wiskundige Informatica

Stageverslag

Michaël Vyverman

Stage promotor: Prof. Dr. P. Dawyndt

Academiejaar 2009–2010

Dankwoord

Deze stage was voor mij een leerrijke en interessante ervaring. Ik zou de stage echter niet hebben kunnen aanvangen zonder de hulp van mijn stage promotor Prof. Dr. Peter Dawyndt die me in contact heeft gebracht met het stage bedrijf Applied Maths N.V.. Professor Dawyndt heeft mij begeleid en voorzien van de nodige informatie over dit opleidingsonderdeel en het onderwerp van deze stage. Verder heeft hij er ook voor gezorgd dat de administratieve kant van dit opleidingsonderdeel vlot verlopen is.

Graag zou ik ook Applied Maths willen bedanken voor het feit dat ik bij hun zes weken lang heb kunnen genieten van de praktijk ervaring die een stage met zich meebrengt. In het bijzonder wil ik ook mijn begeleiders bij Applied Maths bedanken: Paul Vauterin en Hannes Pouseele. Dankzij hun continue inzet heb ik deze stage tot een goed einde kunnen brengen. Zij hebben mijn stage in goede banen geleid en hebben geholpen in het evalueren en verbeteren van de resultaten. Ook kon ik steeds met alle problemen bij hun terecht; vooral voor de technische details van de programmeertaal C++ was dit een must.

Ik wil ook alle mensen bij Applied Maths bedanken voor de manier waarop ik in het bedrijf ben opgenomen en de goede werksfeer die er heerste.

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 Over de stage

In de opleiding Wiskundige Informatica met minor onderzoek bestaat de keuze om ofwel een stage ofwel een seminarie te volgen. Ik heb gekozen voor de stage, aangezien de praktijk ervaring die ik daarmee kon opdoen in een bedrijf een groot voordeel is. Op deze manier heb ik mijn academische kennis kunnen gebruiken in een niet-academisch kader en heb ik kennis gemaakt met een job waarvoor mijn toekomstig diploma zeer geschikt zou zijn.
De stage begon op donderdag 2 juli 2009 en liep ten einde op woensdag 12 augustus 2009. Ik heb in overleg met mijn stage promotor en begeleider bij het stage bedrijf voor deze periode gekozen omdat het zomerreces niet samenvalt met andere opleidingsonderdelen en omdat deze periode het meest geschikt was voor de begeleiders op mijn stageplaats.
Het bedrijf waar ik deze zes weken stage gelopen heb heet Applied Maths N.V. en is een bio-informatica bedrijf gevestigd te Sint-Martens-Latem. Het bedrijf, dat onder andere bekend staat voor zijn software BioNumerics, stelde een project voor waarin er van mij verwacht werd om een literatuurstudie te maken, algoritmen te bestuderen en zelf te implementeren, alsook om een applicatie rond deze algoritmen te schrijven.
Het project betrof het gebruik van suffix bomen en suffix tabellen om herhalingen te vinden in DNA sequenties.
Het project leunt goed aan bij de opleiding Wiskundige Informatica: er is een deel programmeerwerk, alsook een deel algoritmen en data structuren. Voor het biologische onderdeel van het project was Computational Biology (opleidingsonderdeel gevolgd in eerste master) een goede voorbereiding.

1.2 Over de stageplaats

Figuur 1.1: Logo van Applied Maths.

Applied Maths N.V.[1] is een bio-informatica bedrijf. Het werd opgericht in 1992 en heeft ondertussen bekendheid verworven over de hele wereld met zijn software producten. Het hoofdkantoor van Applied Maths is gelegen in Sint-Martens-Latem en een tweede kantoor is gevestigd in de Verenigde Staten van Amerika. Applied Maths is geen groot bedrijf, maar wel een sterk en bloeiend bedrijf[2].
Het bedrijf staat bekend voor het combineren van biologische, technologische en wiskundige expertise. Deze elementen samen zorgen er voor dat het onderzoeksgebied van het bedrijf nauw samenhangt met de opleiding Wiskundige Informatica. Applied Maths heeft succes geboekt in zijn branche dankzij de krachtige en vernieuwende algoritmen. Algoritmen voor patroonherkenning, clusteren, data mining en screening zijn allen verwerkt in de software. De bekendste producten van Applied Maths zijn GelCompar 2 en BioNumerics. GelCompar 2 is een programma voor het analyseren van 2-D patronen. BioNumerics is een volledig bio-informatica pakket voor het opstellen van databases en analyse van biologische informatie. De software van Applied Maths wordt vermeld in vele applicaties en artikels en het bedrijf heeft klanten over de hele wereld, zowel in de academische als in de industriële wereld.

[1] `http://www.applied-maths.com/about`
[2] Biotechnology in Flanders: from science to biobusiness (2003)

Hoofdstuk 2

Technisch verslag

2.1 Stage project

2.1.1 Probleemstelling

Als bio-informatica bedrijf is Applied Maths geïnteresseerd in Next Generation Sequencing. Next Generation Sequencing duidt de recente ontwikkelingen aan in DNA sequenerings apparatuur. DNA sequenering is het proces waarbij in een DNA molecule de samenstelling en volgorde van de vier nucleotiden bepaald wordt waardoor DNA sequenering een belangrijk gegeven in de biologische onderzoekswereld is. De voorbije jaren zijn vele vooruitgangen geboekt in sequenerings apparatuur. De hoeveelheid DNA die in één keer kan gesequeneerd worden is enorm gestegen, waardoor de kost van het sequeneren ook gedaald is. Het proces van sequeneren bestaat eruit eerst het DNA te versnipperen, dan de samenstelling van deze kortere sequenties te bepalen en tot slot de subsequenties te recombineren tot de oorspronkelijke sequentie. Het recombineren van DNA is een algoritmisch probleem en vormt één van de struikelblokken in de nieuwe sequeneringsmethoden.
Een belangrijke factor in het recombineren van DNA zijn de zogehete herhalingen in het DNA: deelsequenties van het DNA die op meerdere plaatsen voor komen. Deze herhalingen bemoeilijken de recombinatie fase indien ze groter zijn dan de woordlengte van de apparatuur[3]. Aangezien de informatie door de sequenerings apparatuur geleverd wordt enorm is, is het belangrijk om herhalingen in DNA sequenties snel en correct te kunnen opsporen. Een manier om heel snel bepaalde zoekopdrachten uit te voeren op sequenties van karakters, hierna 'strings' genoemd, is het gebruik maken van suffix bomen. Een suffix boom is een data structuur die een string verwerkt waarna zoekopdrachten veel sneller kunnen

[3]de lengte van de deelsequenties die apart bepaald worden

worden uitgevoerd. De tijdscomplexiteit van bijvoorbeeld het zoeken naar een deelstring is dan lineair in de lengte van de deelsequentie of substring. Bij de meeste andere zoekalgoritmen is de tijdscomplexiteit van zo een zoekopdracht afhankelijk van de volledige string. Suffix bomen hebben buiten eenvoudige zoekopdrachten nog andere functionaliteiten voor het behandelen van strings en zijn ook interessante data structuren omdat er algoritmen bestaan die efficiënt herhalingen in een string kunnen terugvinden.
De theorie van de suffix bomen wordt echter nog niet overal toegepast en het bedrijf was op het moment van de stage nog niet vertrouwd met de data structuur en zijn mogelijkheden.

2.1.2 Doelstelling

De vraag van Applied Maths was om het potentieel van suffix bomen te onderzoeken: in het algemeen de performantie van de data structuur en in het bijzonder de mogelijkheden voor het zoeken van herhalingen in DNA sequenties. De stage zou bestaan uit drie luiken, waarbij het tweede luik afhing van het succes van het eerste.

1. Een eerste luik van de opdracht bestond uit het maken van een literatuurstudie. Tijdens deze studie was het belangrijk om volgende punten te behandelen:

   - Het onderzoeken van de data structuur suffix bomen. Hierbij aandacht besteden bij de mogelijkheden van de data structuur, de toepassingen en de beperkingen. Op zoek gaan naar recente ontwikkelingen in de theorie omtrent suffix bomen. Ook aandacht besteden aan algoritmen voor constructie en gebruik van suffix bomen.
   - Het zoeken naar een bibliotheek[4] die de data structuur suffix bomen implementeert. De bibliotheek moest ook aan voorwaarden voldoen die verder in het verslag vermeld worden.
   - Een presentatie geven voor collega's waarin de belangrijkste resultaten van de studie vermeld worden.

2. Een tweede deel van de opdracht was afhankelijk van het succes van het eerste. Indien een geschikte bibliotheek in het eerste luik gevonden werd, zou het eerste punt gevolgd worden. Indien dit niet het geval was, dan zou het tweede punt gevolgd worden.

   - Indien een bibliotheek gevonden werd, zou het de bedoeling zijn om met deze bibliotheek te leren werken. Het was de bedoeling om de volledige werking van de bibliotheek te begrijpen, alsook de bibliotheek grondig te testen.

[4] verzameling code die door programma's kan worden gebruikt

- Indien geen geschikte bibliotheek gevonden werd zou zelf een bibliotheek geschreven moeten worden. In dit luik was de doelstelling om klassen en functies te schrijven en algoritmen te implementeren zodat een suffix boom kon worden opgebouwd voor een string en enkele eenvoudige bewerkingen uitgevoerd konden worden op deze implementatie.

3. Een derde en laatste deel van de stage bestond uit het zoeken naar herhalingen. Indien een gevonden bibliotheek dit nog niet (op een efficiënte manier) implementeerde, zou zelf een algoritme geïmplementeerd moeten worden. De soort herhalingen waarin men geïnteresseerd was, zal verderop gedefinieerd worden.

Het uiteindelijke resultaat van de stage zou bestaan uit een presentatie over de literatuurstudie en een bibliotheek die de suffix boom data structuur implementeert. De bibliotheek moet in staat zijn om een suffix boom op te stellen en alle herhalingen waarin men geïnteresseerd was te vinden voor een DNA sequentie van minstens vijf miljoen baseparen in een tijd van ongeveer tien seconden.

2.2 Werkwijze

2.2.1 Literatuurstudie

De literatuurstudie moest voldoende informatie verschaffen over suffix bomen. Dit hield onder andere in de definitie van suffix bomen, details over het gebruik en de constructie van de structuur, de belangrijkste voor- en nadelen, toepassingen en applicaties waarin suffix bomen worden gebruikt en de meest recente ontwikkelingen in de theorie van de suffix bomen. Verder werd gezocht naar een bibliotheek die de data structuur implementeerde en aan bepaalde eisen voldeed.
Ik ben vooral op zoek gegaan naar wetenschappelijke artikels door gebruik te maken van de zoekmachine Google en Google Scholar. Alle gevonden artikels waren vrij beschikbaar. Dit onderdeel van de stage heeft ongeveer een week in beslag genomen. De presentatie op het einde van dit luik werd gegeven op donderdag 9 juli.
In de volgende secties wordt meer informatie gegeven over de resultaten van de studie, in het bijzonder over suffix bomen, een gelijkaardige data structuur: Enhanced Suffix Arrays en over de gevonden bibliotheken.

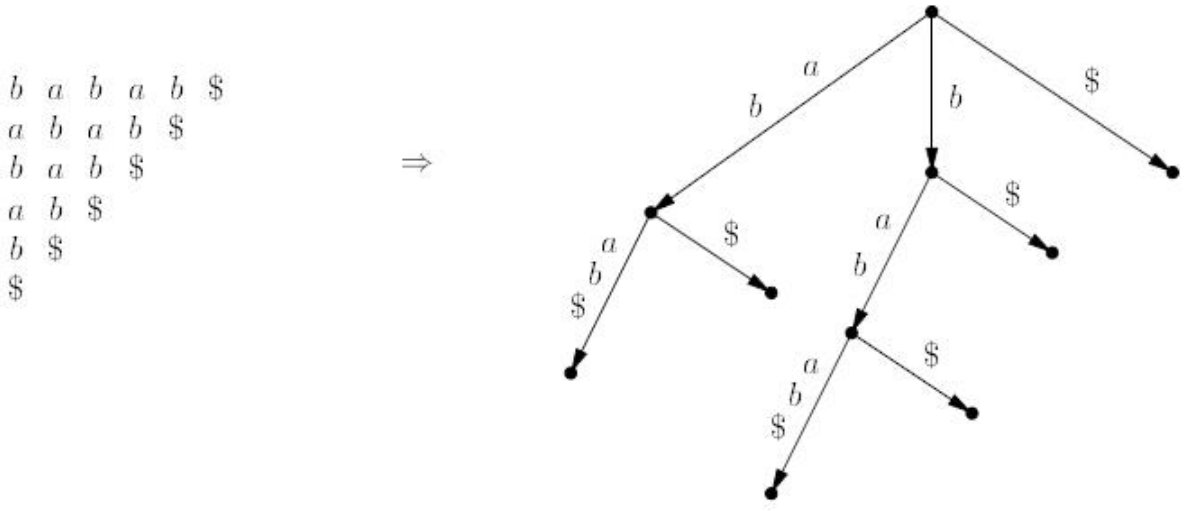
2.2.1.1 Suffix bomen

Suffix bomen zijn, zoals eerder vermeld, data structuren om strings te verwerken. De structuur zelf heeft de vorm van een boom, waarbij de takken gelabeld zijn. De bedoeling is dat elk suffix van de string op een uniek pad kan gelezen worden van wortel tot blad door de labels van de takken aaneen te hechten. De volledige definitie hieronder komt uit het boek [2], dit boek is trouwens een goede introductie tot de theorie van suffix bomen.

Definitie: Een suffix boom T voor een string S bestaande uit m karakters is een gewortelde, gerichte boom met juist m bladeren genummerd 1 tot en met m. Elke inwendige knoop, buiten de wortel, heeft minstens twee kinderen en elke tak is gelabeld met een niet-ledige deelstring van S. Geen enkel paar takken dat uit dezelfde knoop vertrekt heeft een label met hetzelfde beginkarakter. De aaneenhechting van de labels van de takken vormt van wortel tot blad een suffix van S.

Een voorbeeld van een suffix boom is te zien in figuur 2.1, waar de suffix boom voor de string *babab*\$ werd geconstrueerd. Het dollar teken in de string heeft de eigenschap dat het nergens anders voorkomt in de string. De bedoeling van het dollar teken is er voor te zorgen dat de definitie van hierboven kan gebruikt worden; het zorgt er namelijk voor dat elk suffix in een blad eindigt.

Verdere resultaten van de literatuur studie kunnen als volgt samengevat worden:



Figuur 2.1: Suffix boom van de string *babab*\$. De labels van de knopen zijn hier weggelaten. Figuur afkomstig uit [4].

- Voor zoekopdrachten in strings zijn suffix bomen heel snel. De tijdscomplexiteit voor het zoeken naar alle (mogelijk geen) voorkomens van een deelstring in een string is lineair in de lengte van de deelstring en het aantal keer dat de deelstring voorkomt. Daarbij moet wel nog een lineaire tijdscomplexiteit in de lengte van de string zelf bijgerekend worden. Deze laatste is voor het opbouwen van de data structuur. Suffix bomen zijn dus geschikt van zodra meer dan één zoekopdracht uitgevoerd wordt op de string.

- Een nadeel van (de implementatie) van suffix bomen is de grote geheugencomplexiteit. Voor de meeste implementaties kan dit oplopen tot twintig bytes per karakter. Voor strings van enkele miljoenen karakters (de doelstelling) is dit veel.

- De toepassingen van suffix bomen op strings zijn legio. Een aantal van deze toepassingen is volledig uitgelegd in het boek [2]. Als applicatie van suffix bomen kan MUMmer vermeld worden, een programma voor het alligneren van genomen.

- Wat betreft constructie algoritmen is het algoritme van Ukkonen [1] een meilpaal in de geschiedenis van de theorie van suffix bomen. Dit algoritme bereikt een lineaire tijdscomplexiteit en is reeds vaak geïmplementeerd. Een ander algoritme is dat van Kurtz [4]. Dit algoritme bekomt een veel betere geheugencomplexiteit en is momenteel naar mijn mening één van de meest populaire algoritmen.

- Alhoewel suffix bomen onhandelbaar worden voor grote strings door de grote geheugenbehoefte, werd er recent gewerkt aan implementaties van suffix bomen voor het menselijk genoom of voor hele databases van strings.

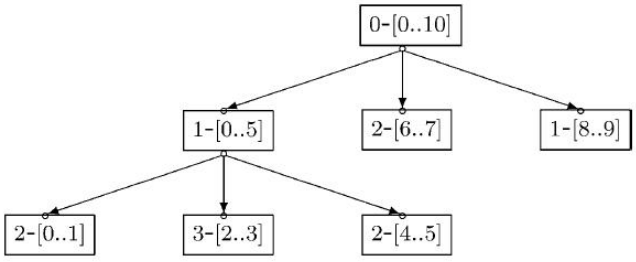
Het nadeel van de grote geheugenconsumptie van suffix bomen heeft mij op het pad gebracht van een andere, gelijkaardige data structuur genaamd Enhanced Suffix Arrays. Deze structuur bezit een veel lagere geheugen complexiteit en heeft toch dezelfde functionaliteit als een suffix boom.

2.2.1.2 Enhanced Suffix Arrays

Enhanced Suffix Arrays steunen op suffix tabellen, dit zijn data structuren die op een soortgelijk idee steunen als een suffix boom, namelijk het indexeren van de suffixen van een string. Op figuur 2.1 (de suffix boom) zien we dat, van links naar rechts in een diepte eerst doorlopen van de boom, de suffixen lexicografisch staan gesorteerd. De suffix tabel bestaat uit een lexicografische ordening van de suffixen, waarbij de positie waarop het suffix begint opgeslaan wordt in de tabel. Het grote voordeel van een suffix tabel ten opzichte

van een suffix boom is de veel kleinere geheugen complexiteit. Een suffix boom kan heel snel meer dan twintig bytes nodig hebben voor elk karakter van de string. Daarentegen heeft een suffix tabel slecht vier bytes nodig per karakter van de string.
Een suffix tabel biedt echter niet dezelfde functionaliteit als een suffix boom. Dezelfde functionaliteit kan wel worden behaald door het toevoegen van extra tabellen, wat leidt tot Enhanced Suffix Arrays. Voortaan zullen we spreken van de ESA structuur om Enhanced Suffix Arrays aan te duiden. Ik heb voor het eerst kennis gemaakt met de ESA structuur in het artikel [6] waar heel grondig wordt uitgelegd hoe een ESA dezelfde functionaliteit kan bieden als een suffix boom en toch een veel kleinere geheugen complexiteit heeft.
De belangrijkste toevoeging aan de suffix tabel is de longest common prefix tabel (lcp tabel). Deze tabel geeft aan hoe lang het gemeenschappelijk prefix is tussen twee opeenvolgende suffixen in de suffix tabel. Aan de hand van deze tabel kan een virtuele suffix boom worden opgesteld en kan deze worden doorlopen zonder hem ooit volledig te construeren. Een voorbeeld van een ESA is te vinden in figuur 2.2. In de figuur is de suffix tabel, lcp tabel, bwt tabel en een tabel die het suffix uitschrijft geconstrueerd voor de string *acaaacatat*\$. De bwt tabel geeft het karakter weer dat één plaats voor het suffix staat. De boom structuur die naast de tabel staat heet de lcp-interval tree en is de eigenlijke suffix boom zonder de bladeren en zonder de labels op de takken. Deze boom wordt geconstrueerd uit de suffix tabel en de lcp tabel.

i	suftab	lcptab	bwttab	$S_{\text{suftab}[i]}$
0	2	0	c	$aaacatat\$$
1	3	2	a	$aacatat\$$
2	0	1		$acaaacatat\$$
3	4	3	a	$acatat\$$
4	6	1	c	$atat\$$
5	8	2	t	$at\$$
6	1	0	a	$caaacatat\$$
7	5	2	a	$catat\$$
8	7	0	a	$tat\$$
9	9	1	a	$t\$$
10	10	0	t	$\$$



Figuur 2.2: Enhanced Suffix Array en lcp-interval tree van de string *acaaacatat*\$. Figuur afkomstig uit [6].

De ESA structuur bevat een zekere modulariteit. Voor de volledige functionaliteit van suffix bomen moeten nog meerdere tabellen geconstrueerd worden, maar voor andere opdrachten zijn bovenstaande tabellen voldoende. Ook voor het zoeken naar enkele soorten herhalingen zijn bovenstaande tabellen voldoende. Door deze modulariteit kan op zowel constructietijd als geheugen bespaard worden.
Na overleg met mijn begeleiders werd beslist dat verdere implementatie en onderzoek op Enhanced Suffix Arrays zou worden uitgevoerd in plaats van suffix bomen. Deze beslissing werd gemaakt omdat ESA de meeste voordelen van suffix bomen behouden en de geheugen kost van suffix bomen minimaliseren. Ook zijn suffix tabellen een eenvoudiger concept en eenvoudiger te implementeren.

2.2.1.3 Bibliotheken voor suffix bomen

Als laatste opdracht van de literatuurstudie werd gevraagd om te zoeken naar een bibliotheek die de data structuur, suffix bomen, implementeert. Indien zo een bibliotheek gevonden werd, dan zou tijdens het verdere verloop van de stage de nadruk liggen op het zoeken naar herhalingen in DNA. Indien geen geschikte bibliotheek gevonden werd zou in het volgende deel van de stage meer de nadruk liggen op het zelf implementeren van een suffix boom bibliotheek. Aangezien op het moment dat deze studie plaatsvond de beslissing om te schakelen naar Enhanced Suffix Arrays nog niet gemaakt was, werd alleen gezocht naar bibliotheken die suffix bomen implementeren.
De bibliotheek moest aan bepaalde criteria voldoen. Er waren harde eisen die zeker vervuld moesten zijn en zachte eisen die een bepaalde bibliotheek een voordeel konden opleveren ten opzichte van een andere. De harde voorwaarden waren:

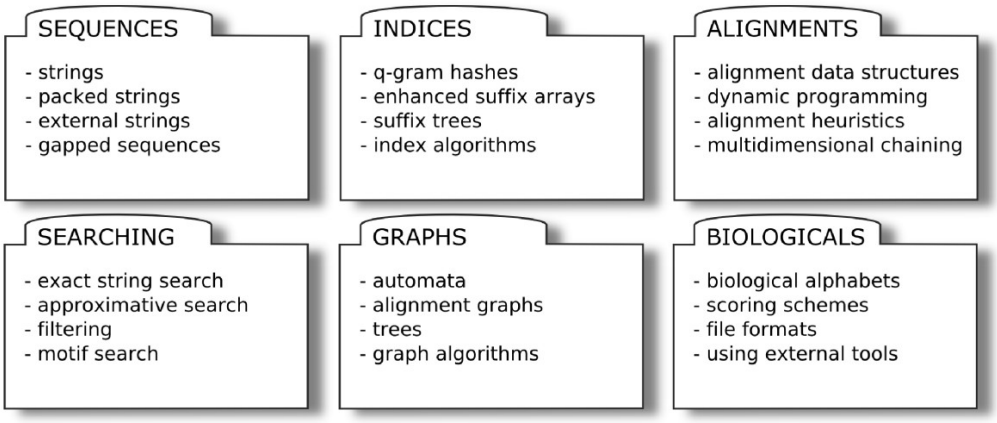
- De bibliotheek moest in C++ geschreven zijn of desnoods in C. Deze eis volgde uit het feit dat C++ de standaardtaal is in het bedrijf en verdere uitbreidingen van de bibliotheek ook in C++ geschreven zouden worden.

- De licentie waaronder de bibliotheek viel moest commercieel gebruik toelaten. Hierin verschilt de stage met een project in een academisch kader.

- Uiteraard moest de bibliotheek werken en ook de doelstelling halen van vijf miljoen baseparen.

Verdere eigenschappen van een bibliotheek die de keuze konden beïnvloeden bestonden uit onder andere: de kwaliteit van de bijgeleverde documentatie, het al dan niet platform onafhankelijk zijn van de bibliotheek, de uitbreidbaarheid en leesbaarheid van de code en

functies die reeds het zoeken naar herhalingen implementeren. Verder waren referenties naar en goede commentaren op de bibliotheek een pluspunt.
De resultaten van een eerste studie leverden zeven bibliotheken op die aan de strikte eisen voldeden, maar slechts één daarvan was geprogrammeerd in C++. Verder onderzoek zorgde voor een schifting in de bibliotheken naar gelang de geïmplementeerde algoritmen en de bijgeleverde informatie. Uiteindelijk bleven drie bibliotheken over waarvan degene die in C++ geschreven was een grote voorsprong op de andere had.
De bibliotheek die de test fase haalde was de SeqAn bibliotheek[5]. De bibliotheek wordt beschreven in [7]. De Seqan bibliotheek is zoals reeds vermeld in C++ geschreven, valt onder de Lesser GNU Public License die commercieel gebruik toelaat en is heel goed gedocumenteerd. De bibliotheek is recent gebruikt in applicaties en bevat niet alleen een implementatie van suffix bomen, maar ook een implementatie van Enhanced Suffix Arrays. De bibliotheek biedt ook andere functionaliteiten aan die beschreven staan in figuur 2.3.
Tijdens enkele tests kwamen echter bugs aan het licht in de bibliotheek. Ik heb na lang



Figuur 2.3: Functionaliteiten van de SeqAn bibliotheek. Figuur afkomstig van [7].

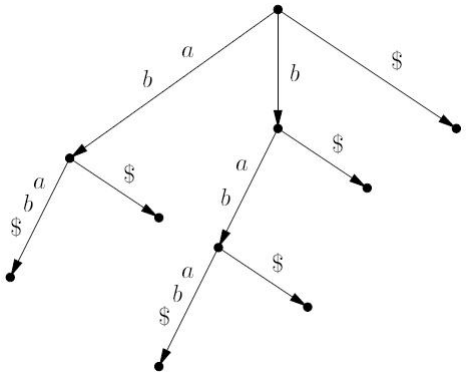
zoeken de bugs kunnen verwijderen uit de code, maar het vertrouwen in de bibliotheek was door de aanwezigheid van de bugs geschaad.
Uiteindelijk werd besloten om de SeqAn bibliotheek niet verder te gebruiken. Vooral omdat bugs aanwezig waren in een cruciaal gedeelte van de code en de code vrij onoverzichtelijk was waardoor verdere bugfixing moeilijk zou zijn. De beslissing viel op het zelf schrijven van een bibliotheek. Dit had als voordeel dat er volledige controle bestond over de inhoud en de werking van de bibliotheek.

[5]Deze is te vinden op de website `http://www.seqan.de`.

2.2.2 Programmeren van een Enhanced Suffix Array bibliotheek

Aangezien in het vorige luik van de stage werd beslist om zelf een bibliotheek op te stellen, werd dit de taak in dit onderdeel van de stage. De bibliotheek moest in staat zijn om alle exacte maximaal herhaalde paren in een gegeven string te vinden. Deze herhalingen zijn maximaal in de zin dat door ze uit te breiden naar links of rechts het paar een verschillend teken bevat op die posities. Deze definitie geldt uiteraard alleen voor paren. Om deze herhalingen te bepalen moest eerst de suffix en de lcp tabel opgesteld worden voor de gegeven string. Verder moest de lcp-interval boom doorlopen worden van onder naar boven (van blad naar wortel). Tot slot moest voor elke knoop uit de boom de maximaal herhaalde paren berekend worden.

Voor deze ontwikkelingsfase werd geprogrammeerd in C++, waarbij gebruik gemaakt



Figuur 2.4: Dezelfde suffix boom als figuur 2.1. Alle maximale herhalingen zijn af te lezen van wortel tot een inwendige knoop. De bladeren vormen dan de paren. Niet elk paar is echter maximaal.

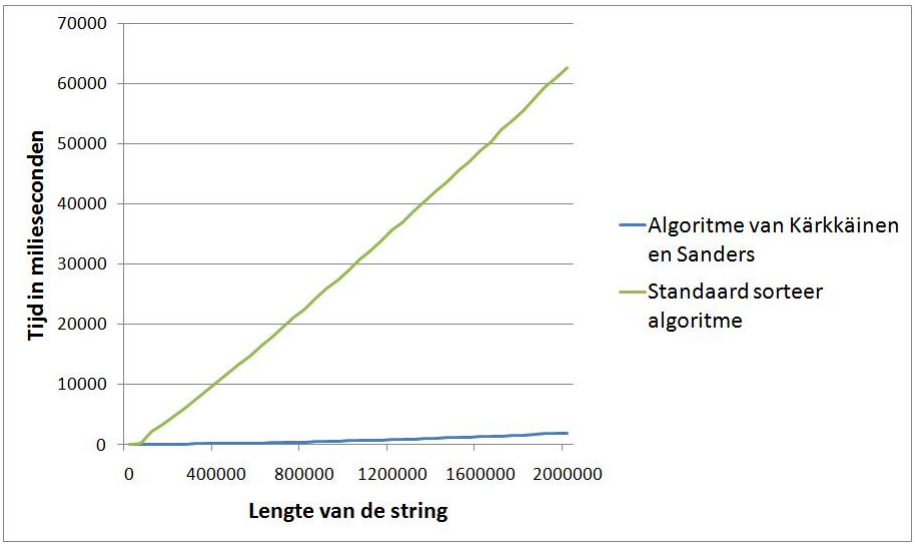
werd van Microsoft Visual Studio 2008. Dit onderdeel van de stage duurde ongeveer twee weken.

In eerste instantie werd de data structuur vastgelegd, waarna een werkende (inefficiënte) versie van het programma werd gemaakt. Pas nadat een volledig werkend programma ontwikkeld was werd er gewerkt aan de doelstelling van vijf miljoen baseparen.

De eerste dagen heb ik een spoedcursus C++ gekregen, aangezien deze programmeertaal volledig nieuw voor mij was. Tot dan toe was Java de enige programmeertaal die ik beheerste. Ondertussen werd ook de data structuur vastgelegd. Mijn begeleider, Hannes Pouseele, schreef zelf een implementatie van een tabel die verder gebruikt werd voor de ESA structuur. Verder werd er ook beslist dat een volgorde op de karakters van een string gedefinieerd moest worden, wat aanleiding gaf tot een 'alfabet' klasse.

Aangezien een suffix tabel de suffixen van een string lexicografisch ordent, werd als een con-

structie algoritme het sorteer algoritme gekozen uit de standaard bibliotheek van C++. De lcp tabel geeft tussen twee opeenvolgende suffixen in de suffix tabel aan wat hun gemeenschappelijke prefix is. Dit werd berekend door elk karakter van elk opeenvolgend paar suffixen te vergelijken tot een verschillend karakter gevonden werd.
De lcp-interval boom is een boom structuur waarin de knopen van de boom intervallen zijn in de lcp tabel. De waarden van de elementen in zulke intervallen moeten allen een minimum waarde overschrijden. De volledige boom waarvan een voorbeeld is te zien in figuur 2.2, is gelijkaardig aan een suffix boom waarbij de bladeren verwijderd zijn. Om deze lcp interval boom te doorlopen werd een algoritme gebruikt uit het artikel [6]. Uit hetzelfde artikel werd het algoritme geïmplementeerd dat zoekt naar exacte maximale herhalingen in de string.
De eerste versie van de bibliotheek kon de gezochte herhalingen correct berekenen. Zoals verwacht werden de doelstellingen met deze eenvoudige algoritmen niet behaald. Voor willekeurige strings met een lengte die tien keer kleiner waren dan de doelstelling van vijf miljoen baseparen had de bibliotheek al meer dan een minuut constructie tijd nodig. Tijdens de testen bleek ook dat herhaalde paren zoeken heel tijdrovend is. Om dit te verhelpen werd een minimum grens ingesteld op de lengte van de gezochte herhalingen, waardoor het aantal gezochte herhalingen alsook de looptijd van het programma drastisch daalde.



Figuur 2.5: Grafiek die het verschil aangeeft tussen constructie algoritmen voor de suffix tabel. Als string is de ecoli sequentie gekozen (NC_000913) waarvan de eerste n karakters zijn genomen; n varieert op de $x-$as. De constructie tijd is te vinden op de $y-$as.

In een tweede fase werd naar nieuwe algoritmen gezocht die ervoor zouden zorgen dat de bibliotheek aan de doelstelling voldeed. Voor het construeren van de suffix tabel werd het algoritme van Kärkkäinen et al. [5] geïmplementeerd en voor het construeren van de lcp tabel werd het algoritme van Kasai [3] geïmplementeerd. Voor het doorlopen van de lcp-interval boom en het zoeken naar de herhaalde paren werden de bovenstaande algoritmen behouden. De nieuwe algoritmen waren vrij eenvoudig te implementeren en bevatten geen ondoorgrondelijke stappen. De nieuwe resultaten zorgden voor een heel grote tijdswinst en de doelstelling werd behaald.
De bibliotheek werd in deze fase ook gecontroleerd op memory leaks en de bottlenecks werden opgespoord dankzij het programma Automated QA. Na de nodige verbeteringen te hebben aangebracht kon de bibliotheek voor een willekeurige string, met alfabet grootte vier en lengte vijf miljoen karakters, de exacte maximaal herhaalde paren met minimumlengte tien berekend worden in tien tot dertien seconden. Het verschil tussen de twee suffix tabel constructie algoritmen is terug te vinden in figuur 2.5.
Door het succes van de nieuwe algoritmen werd mij gevraagd om een presentatie te geven waarin ik de werking van alle gebruikte algoritmen zou verklaren.

2.2.3 Herhalingen in DNA sequenties

In dit derde deel van de stage werd de nadruk gelegd op de herhalingen in DNA sequenties. Dit deel omvatte het vinden van een geschikte manier om exacte herhalingen uit te breiden naar niet exacte herhalingen, een manier om herhalingen te presenteren en het uitbreiden van de bestaande bibliotheek met een algoritme die dit implementeert.
Dit luik duurde ongeveer twee weken, waarin een literatuurstudie over herhalingen werd gemaakt en de bibliotheek uitgebreid werd met de nodige algoritmen.
In dit deel werd de uitbreiding gemaakt naar herhaalde paren die een zekere homologie hebben. Dit is het percentage van hun lengte die bestaat uit overeenkomstige karakters. De uitbreiding waarbij ook inserties en deleties beschouwd zouden worden werd niet gemaakt aangezien dit de bibliotheek meteen zou vertragen. Een uitbreiding met indels zou waarschijnlijk een kwadratisch algoritme vergen, gelijkaardig aan het Needleman-Wunsch algoritme voor globale alignering van sequenties.
Het algoritme zou moeten steunen op het uitbreiden van exact herhaalde paren, naar grotere herhaalde paren met een zekere homologie. De literatuurstudie was geen succes. Het idee achter het uiteindelijke algoritme kwam van mijn begeleider Paul Vauterin. Zijn voorstel was om twee grenzen te gebruiken: een lage homologie grens die als 'zoekruimte' gebruikt werd en een hoge homologie grens die de uiteindelijke strikte grens vormde. Het

algoritme dat uit dit idee voortvloeide werkte heel goed, zo bleek uit de tests.
Deze tests vormden de laatste stap van dit luik van het project. Tijdens deze fase werd de bibliotheek ook uitgebreid met enkele test klassen. Uit de tests bleek dat het extra algoritme bijna geen vertraging opleverde en zelf voor een verbetering in geheugenkost zorgde doordat herhalingen die reeds binnen een uitgebreide regio vielen niet meer werden bewaard. Uit de tests bleek ook dat de parameters voor het zoeken naar herhalingen best lager werden gekozen dan theoretisch verwacht.

2.2.4 Optimaliseren van de code

Aangezien de doelstellingen reeds bereikt waren, werd in de laatste week aandacht besteed aan optimalisatie en vormgeving van de code. De bibliotheek werd aan een inspectie ondergaan van een collega. Tijdens deze controle kwamen vele stijlfouten aan het licht. Vooral in zake abstractie en afhankelijkheid kon de code nog veel verbeterd worden. Deze optimalisaties werden echter nog niet van mij verwacht en de fouten werden gezien als een gebrek aan ervaring op dit niveau van programmeren. De laatste week heb ik getracht om zoveel mogelijk de code te verbeteren en te becommentariëren.

2.3 Discussie

In de bibliotheek werd gebruik gemaakt van Enhanced Suffix Arrays in plaats van suffix bomen. De modulariteit van de ESA is goed tot zijn recht gekomen in dit project aangezien slechts twee tabellen nodig waren voor de gevraagde applicatie en er zeker vier nodig zijn om de gehele functionaliteit van suffix bomen te verkrijgen.
Dankzij de beslissing om zelf een ESA bibliotheek op te stellen is de volledige werking van de bibliotheek duidelijk. Tijdens de stage werd er regelmatig overleg gepleegd, waardoor de werking van de bibliotheek ook duidelijk is voor mijn begeleiders. Dankzij de presentatie over de constructie algoritmen op het einde van het tweede luik is de werking van deze algoritmen ook duidelijk.
De gebruikte data structuren en algoritmen zijn niet optimaal, maar dit was ook niet vereist. Verdere optimalisatie werd niet gevraagd aangezien er vaak in een commerciële omgeving een trade-off bestaat tussen de geleverde inspanning en de gewenste graad van optimalisatie.
Tijdens het project heb ik de kans gehad om het nut van bepaalde eigenschappen van 'goed' programmeren in praktijk te ondervinden. Begrippen zoals abstractie, afhankelijkheid, cohesie en modularisatie komen voor in een allereerste cursus programmeren, maar nu werd

hun nut bewezen. Door het abstraheren van de data structuren kon een verandering in deze structuren snel en eenvoudig worden doorgevoerd. Door het beperken van de afhankelijkheid konden sommige klassen en functies eenvoudiger uitgebreid worden. Vooral naar het einde van het project toe werd de noodzaak van deze begrippen getoond.
Ook in het debuggen en het opsporen van bottlenecks heb ik ervaring opgedaan. Zo heb ik ondervonden dat zelfs eenvoudige bewerkingen tijdrovend kunnen worden indien ze vijf miljoen keer worden uitgevoerd.

2.4 Resultaat

De doelstelling van het project is zeker gehaald. Allereerst heeft de literatuurstudie een verzameling artikels opgeleverd waarin de gebruikte algoritmen werden gepresenteerd, maar ook artikels die kunnen gebruikt worden bij verdere uitbreidingen van de bibliotheek en toepassingen van Enhanced Suffix Arrays.
Verder heb ik in de loop van de stage twee presentaties gegeven, waardoor mijn collega's nu ook iets meer weten over suffix bomen en tabellen.
Tot slot is er de bibliotheek zelf die in C++ geschreven is en aan de gestelde vereisten voldoet. De functionaliteit van de bibliotheek bestaat uit het opbouwen van de suffix en lcp tabel voor een string en het vinden van alle (in)exacte herhaalde paren in deze string.

Hoofdstuk 3

Conclusie

3.1 Conclusie en toekomstig werk

In de vorige sectie heb ik reeds vermeld wat het resultaat was van het project. Voor Applied Maths betekent dit resultaat dat ze de werking van suffix structuren gezien hebben, waaronder de exacte werking van de algoritmen en de voordelen van het werken met de ESA structuur. Mijn begeleiders beschouwden het project geslaagd. Ook vonden ze het concept van stage, wat nieuw voor hen was, geslaagd en zullen ze in de toekomst hoogst waarschijnlijk nog stagairs verwelkomen.
Deze stage was voor mij ook een meerwaarde. Ik heb de kans gekregen om buiten een academisch kader mijn kennis toe te passen en ervaring opgedaan in een bedrijfsomgeving. Het project was een goede toets voor mijn praktische kennis en hiermee heb ik mijn sterke en minder sterke punten leren kennen. Daarbovenop heb ik kennis gemaakt met een nieuwe programmeertaal en met enkele nieuwe tools voor het ontwikkelen van software.
Applied Maths zal mijn werk integreren in een groter project. Een Enhanced suffix Array zal slechts één van de data structuren zijn in dit groter geheel van applicaties rond strings (DNA sequenties). Dit neemt niet weg dat de Enhanced Suffix Array zelf niet zal worden uitgebreid. Een uitbreiding die meerdere strings tegelijkertijd verwerkt of die toelaat een string dynamisch aan te passen in de data structuur zijn mogelijke uitbreidingen.
Ikzelf zal ook nog verder de theorie rond suffix bomen en Enhanced Suffix Arrays onderzoeken. Het project bij Applied Maths bevatte slechts enkele toepassingen van deze structuren, maar dit was reeds genoeg om hun nut te bewijzen.

3.2 Persoonlijke evaluatie van de stage

In het algemeen was het vak stage zeker de moeite waard. Tijdens de zes weken die ik bij Applied Maths gewerkt heb, heb ik veel bijgeleerd dat zowel mijn opleiding als mijn latere werk ten goede zal komen. Ik wil hier nog even kort mijn persoonlijke mening over het vak vermelden.

Wat betreft de organisatie van dit opleidingsonderdeel, vond ik dat er meer informatie mocht verschaft worden in het begin van het academie jaar. Een infosessie waarin uitgelegd werd wat de stage precies inhoudt, hoe een bedrijf gecontacteerd kan worden of bij wie we met onze vragen terecht kunnen, zou een must zijn. Van zodra ik in contact kwam met Professor Peter Dawyndt, stage coördinator en mijn stage promotor, werden veel van mijn vragen beantwoord en Professor Dawyndt heeft mij in contact gebracht met Applied Maths.

Wat betreft de verdere organisatie heb ik ook gemerkt dat dit opleidingsonderdeel nog in zijn kinderschoenen stond. Professor Dawyndt heeft zich steeds ingespannen om de administratie in goede banen te leiden.

Wat betreft de periode waarin de stage verliep, ben ik er nog steeds van overtuigd dat het zomerreces tussen het eerste en het tweede Master jaar de beste periode voor de stage is. Mijn keuze voor een periode van zes weken in plaats van vier weken was op aanraden van Professor Dawyndt. De periode is vrij lang, maar de extra weken waren voor mij even leerrijk en nuttig als de eerste vier weken.

Het bedrijf Applied Maths is zeker aan te raden als stage bedrijf. Het bedrijf kan projecten aanbieden die nauw aansluiten bij de opleiding en uitdagend en verijkend zijn. Mijn begeleiders bij het bedrijf hebben mij zeer goed begeleid en ook van andere collega's heb ik iets bijgeleerd. Daarbovenop heerst er in het bedrijf een zeer goede werksfeer.

Het project bevatte enkele uitdagingen die vooral te maken hadden met enkele leemtes in mijn opleiding tot nu toe. Het is dan ook handig om op tijd in contact te komen met het stage bedrijf zodat sommige vaardigheden kunnen worden opgefrist of aangeleerd.

Als conclusie mag gezegd worden dat dit stage vak één van de meest interessante onderdelen van mijn opleiding was.

Bibliografie

[1] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, (14):249–260, 1995.

[2] D. Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge university press, 32 Avenue of the Americas, New York, NY 10013-2473, USA, 11th edition, 1997.

[3] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. *Proc. 12th Symposium on Combinatorial Pattern Matching (CPM 01) Springer-Verlag LNCS*, (2089):181–192, 2001.

[4] R. Giegerich, S. Kurtz, and J. Stoye. Efficient implementation of lazy suffix trees. *Software - practice and experience*, (33):1035–1049, 2003.

[5] J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. *In Proceedings of the 30th International Conference on Automata Languages and Programming*; *Lecture Notes in Computer Science*; *Springer-Verlag*, (2719):943–955, 2003.

[6] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, (2):53–86, 2004.

[7] A. Döring, D. Weese, T. Rausch, and K. Reinert. Seqan an efficient, generic c++ library for sequence analysis. *BMC Bioinformatics*, 2008.